1 ImageGear Professional v18.1 - User Assistance Dashboard

1.1 Copyright Information

1.2 User Guide

The ImageGear Professional for Mac User Guide provides the following information:

- The Introduction provides information about ImageGear features, improvements, and component structure.
- The Installing, Licensing, and Distributing chapter provides system requirements information as well as installation instructions. It also describes the structure of installed ImageGear files and the ImageGear Licensing process.
- The chapter describes how to incorporate ImageGear's imaging capabilities into your own applications by using the sample source imaging programs provided with your toolkit.
- The Using ImageGear chapter provides information about how to call an ImageGear function, including setting up the variables and structures passed as arguments, and details which functions to use to perform a variety of imaging tasks. It also explains how to work with some ImageGear component API.
- The File Format Reference chapter provides detailed information for each file format supported by ImageGear, along with the corresponding control parameters, supported features and compressions, plus "at-a-glance" tables that describe the features supported by each format, providing guidelines on which formats may be most suitable when speed, storage space, or similar considerations are important.
- The Appendices/General Reference chapter provides general reference information.

Refer to the API Reference Guide for detailed information about the ImageGear API.

1.2.1 Introduction

This Introduction provides information about the following:

- What's New in ImageGear Professional
- ImageGear Overview
  - High Speed Display
  - Image Loading and Saving
  - Printing
  - Image Processing
  - Pixel Access
  - ROI Processing
  - Format Conversion
  - Compression
  - Transitions
  - Native CMYK Support
  - Database
  - Supported Formats and Compressions
  - ImageGear Samples
- About Customer Support

1.2.1.1 What's New in ImageGear Professional

ImageGear Professional v18.1

ImageGear Professional v18.1 for Mac introduces the following new features:

- Core, LZW, Medical and PDF components are now supported on the following Mac OS X platforms:
  - Mac OS X v10.7 Lion (64-bit)
  - OS X v10.8 Mountain Lion (64-bit)
  - OS X v10.9 Maverics (64-bit)
- Improved licensing tools for managing evaluation, toolkit, and runtime deployment licenses to simplify user experience
- Updated PDF support
- New Medical/DICOM support

1.2.1.2 ImageGear Overview

ImageGear® Professional for Mac is the most advanced way to create, control, and deliver more secure, high-quality imaging applications. ImageGear allows you to add powerful imaging capabilities to your applications. ImageGear supports the most commonly used graphics file formats, providing complete compatibility when developing applications across multiple platforms, or when developing for users who have a variety of target systems.

This section provides overview information about the following ImageGear features:

- High Speed Display
- Image Loading and Saving
- Printing
- Image Processing
- Pixel Access
- ROI Processing
- Format Conversion
- Compression
- Transitions
- Native CMYK Support
- Database
- Supported Formats and Compressions
- ImageGear Samples

1.2.1.2.1 High Speed Display

ImageGear Professional gives you complete control over how your application displays each of its images. Among the attributes you can set on an image-by-image basis are:

- Auto-color reduction for low color modes
- High-quality display for all video modes
- Contrast/Brightness
- Display effects (wipes, blocks, etc.)
- Color reduction (several types)
- Transparency
- Gamma correction
- Dithering
- Rotation
- The portion of the image to display (the array of pixels within the DIB bitmap is called the Image Rectangle)
- How to fit the Image Rectangle to the Device Rectangle
- The background fill pattern and color to use (for any area of the Device Rectangle left vacant by the image)
- Region within the display area the image is to be displayed (this area is called the Device Rectangle)
- Center image in window
- Fit to width, height, window
- Precision scrolling
- Sub-region display
- Auto-aspect ratio
- Preserve black and preserve white display for 1-bit images
- Sub-second display rotation supported
- Faster image display
- Anti-alias display
- Huge image display capability
- Merge 2 images during display
- 4x faster scale-to-gray
- Center, zoom, or scroll a displayed image from within your application
- Use an image's LUTs (Look-Up Tables) to translate the palette to another set of colors

All of the above display attributes affect the display only. They do not alter either the image bitmap or the color palette in the DIB.

See Also:

Loading Images

Saving Images

Displaying Images

Thread Safety

Understanding Display Options

Color Reduction

Color Promotion

Contrast Alteration

Inquiring Format Filters for Supported Features

Using Color Profile Manager

Global Control Parameters

Grayscale Look-Up Tables

Working with Multi-Page Documents

Stripped Images

Tiled Images

1.2.1.2.2 Image Loading and Saving

ImageGear Professional supports over 200 raster file formats.

See Also:

Loading Images

Saving Images

Displaying Images

File Format Reference

1.2.1.2.3 Printing

ImageGear Professional provides the following print capabilities:

- Print to any printer with complete control.
- Auto-color reduction for high-quality printing.
- Single- or multi-page printing.
- Automatic sizing to full-, half-, quarter-, eighth-, and sixteenth-page, with auto page centering, or specific placement.
- Print multiple images to a single page.
- Print images to specified location and at specified size.

See Also:

Printing Images

1.2.1.2.4 Image Processing

ImageGear Professional provides the following image processing capabilities:

- Region of interest (ROI) support for basic rectangles as well as ellipses, polygons, and other shapes.
- Complete color space support, including color space conversions, color separation and combination, support for any color space found in any of the 200 supported formats, and support for color spaces such as CMYK without conversion to RGB.
- Color reduction that maximizes quality and minimizes size.
- Encryption and decryption of an entire image or any part of an image.
- Matrix convolutions of any size with pre-defined or user-supplied matrix values.
- Special effects.
- Automatic image correction.
- Intelligent re-sizing.

See Also:

Image Processing

Core Component API Function Reference

1.2.1.2.5 Pixel Access

ImageGear Professional is equipped with several functions that will allow you to get and set the values of individual pixels, rows or columns of pixels, and rectangular groups of pixels. This family of functions is referred to as the "pixel access" functions. For each kind of pixel access, you can obtain the value(s) of a pixel or group of pixels, or set the value(s) of a pixel or group of pixels.

See Also:

Pixel Access Operations

1.2.1.2.6 ROI Processing

ImageGear Professional provides the following Region of Interest (ROI) functionality:

- Specify rectangular ROI for nearly all image processing functions.
- Specify arbitrary ROI for most image processing functions. Functions included to create certain shape types such as ellipse, polygon, and freehand.
- Create a 1-bit mask image for identifying which pixels to include/exclude from image processing algorithms.

See Also:

Image Processing

Core Component API Function Reference

1.2.1.2.7 Format Conversion

ImageGear Professional supports over 200 raster file formats. To convert a file from one format to another, ImageGear allows you to save the original file to a different format by setting the nFormatType parameter to the appropriate value in the saving function. For more information on converting images, see the section Saving Images.

See Also:

File Format Reference

ImageGear Supported File Formats Reference

ImageGear Supported Bit Depths

1.2.1.2.8 Compression

ImageGear Professional supports most of the industry-standard compression algorithms.

See Also:

ImageGear Supported Compressions Reference

ImageGear Supported File Formats Reference

1.2.1.2.9 Transitions

ImageGear Professional provides the ability to specify the type of transition to use from one image to another. This is useful for slide shows or in any other case where the image itself is the focus. For this type of product, ImageGear provides a set of functions for migrating from one image to another.

The transition support in ImageGear includes 29 types of transitions with control over the granularity and speed. Granularity refers to the size of the object used or the smoothness of the transition. The speed is the total time used to transition from one image to another. In addition, all other display parameters are available for controlling the transitions.

See Also:

Image Transformation

Image Analysis

Blending and Combining Images

Image Correction

Image Maintenance

1.2.1.2.10 Native CMYK Support

ImageGear Professional Native CMYK support entails the following:

- Images stored in CMYK format can be loaded into ImageGear without being converted to RGB to ensure the original color information is maintained.
- Images can be saved to formats that support CMYK color space.
- The majority of image processing, image access, display, and other functions work with native CMYK image data.

1.2.1.2.11 Database

ImageGear Professional provides the following database functionality:

- Load images from memory in more than 200 formats.
- Load images from file with any offset for images embedded in a database.
- Decode images using various compression algorithms without specific format headers.
- Import/export images from/to various types of memory formats.
- Save images to files at specified offsets.

1.2.1.2.12 Supported Formats and Compressions

ImageGear Professional supports the most commonly used graphics file formats with different compressions. The ImageGear-supported file formats are described in detail in the File Format Reference chapter.

1.2.1.2.13 ImageGear Samples

Your ImageGear Professional toolkit also contains a directory with sample imaging application programs and images. You may copy and modify them as needed. You can also use them as templates for developing your own applications. The sample images may also be used for any purpose, such as testing your applications as you develop them.

See ImageGear Samples for a complete list of the samples available, along with their descriptions, and installed location.

1.2.1.3 About Customer Support

If you are unable to find an answer to your questions in the help, refer to the Release Notes, which provide release-specific information, including changes made to the product since the last update. If you need additional assistance, please read the procedures below:

- Double check this User Guide. In particular, refer to the following chapters: and Using ImageGear. These chapters contain a great deal of information on both programming your application and identifying image problems.
- Take a look at the sample programs included with your product.
- Visit the Support Page on the Accusoft web site or call Accusoft at 813-875-7575.
- Accusoft.com provides extensive product information, including:
  - Product Information and Specifications
  - Product Downloads
  - Customer Support
  - Demonstrations and Tutorials
  - On-line Documentation

If you still need technical support assistance, please refer to the Software Support and Maintenance Policy on the Accusoft web site.

1.2.2 Installing, Licensing, and Distributing ImageGear

This section provides information about how to install/uninstall ImageGear Professional for Mac in the following sections:

1.2.2.1 Minimum Requirements

Before installing ImageGear Professional for Mac, make sure that your computer system meets the minimum requirements detailed in this section.

Supported Hardware:

- x86-64 Apple Mac

Supported Operating Systems:

- Mac OS X v10.7 Lion (64-bit)
- OS X v10.8 Mountain Lion (64-bit)
- OS X v10.9 Maverics (64-bit)

Java Requirement for Licensing Tools:

- JDK: Oracle Java SE Development Kit 1.7 or later (to run License Manager and Server Licensing Utility)

Please make sure that correct version of Java is used. To check the installed Java version run the following command in terminal:

```
java -version
```

1.2.2.2 Installing ImageGear

To install ImageGear, download an electronic version from www.accusoft.com. Please contact Accusoft at 813-875-7575 for instructions on downloading your specific version of ImageGear Professional for Mac.

The name of the ImageGear installation package for Mac OS X is **ImageGearPro18.1.1-Mac64.dmg (OS X 64-bit platform)**

Please see one of the following sections below for details on installing ImageGear:

- Automated Installation
- Manual Installation

Automated Installation

1. Install the latest Oracle Java SDK from www.java.com, which is required for the licensing tools.
2. Mount the installation file **ImageGearPro18.1.1-Mac64.dmg** as a volume within the Finder.
3. Start the installation process by double-clicking **ImageGearPro18.1.1-Mac64.pkg**.

   The installation script will search for and modify the current user's profile files to add ImageGear's environment variable and attempt to run the Accusoft License Manager at the end of the installation.

Manual Installation

1. Mount the installation file **ImageGearPro18.1.1-Mac64.dmg** as a volume within the Finder.
2. Start the installation process by double-clicking **ImageGearPro18.1.1-Mac64.pkg**.
3. In order for ImageGear to work, there must be a license file installed. The license key is kept in the file with the name **accusoft.<solution name>.<version specification>.imagegear**. The ImageGear installation comes with the predefined solution name 'Accusoft' and version specification '1-21-18' for Mac 64-bit platform.

   If the installation script failed to modify the current user's profile files to add ImageGear's environment variable or to run the Accusoft License Manager at the end of the installation, please proceed with the following manual steps:

   a. The variable **IMAGE_GEAR_LICENSE_DIR** has to be defined and should contain the path to the user's licensing location: **$HOME/Accusoft/ImageGear18/Licensing**. If the **IMAGE_GEAR_LICENSE_DIR** variable is not defined, ImageGear will look for the license file in **/Library/Frameworks/ImageGear18.framework** directory.
   b. Install the latest Oracle Java SDK from www.java.com to be able to run the Accusoft License Manager using the command line **/Accusoft/ImageGear18/Licensing/LicenseManager/runLicenseManager** script. Alternatively, you can install the latest Oracle Java run-time component from www.java.com to run the Accusoft License Manager using **/Accusoft/ImageGear18/Licensing/LicenseManager/LicenseManager.jar**.

   Please refer to ImageGear Licensing for additional details about how to acquire a license key.

1.2.2.3 Directory Structure

The ImageGear for Mac OS X installation will create the following directory structure, assuming that the home directory is the install directory:

/Accusoft/

/Accusoft/ImageGear18/

/Accusoft/ImageGear18/Bin/

/Accusoft/ImageGear18/Documentation/

/Accusoft/ImageGear18/Documentation/HTML

/Accusoft/ImageGear18/Documentation/Release Notes

/Accusoft/ImageGear18/Licensing/

/Accusoft/ImageGear18/Licensing/Deployment

/Accusoft/ImageGear18/Licensing/LicenseManager

/Accusoft/ImageGear18/Samples

/Accusoft/ImageGear18/Samples/Xcode

/Accusoft/ImageGear18/Samples/Xcode/ImageGearDemo

/Library/Frameworks/ImageGear18.framework

$HOME/Accusoft/ImageGear18/Licensing/

1.2.2.3.1 Description of Installed Files

All the files installed using the install procedure as described above will be installed in the directory structure shown above. Depending on the version of the library you have purchased, the file names will change, as well as the file sizes and usage requirements. Also included will be one or more sample programs installed in the sub-folder named "Samples". These sample programs can be used as guides or examples of how to use the libraries in your applications. You may cut and paste freely from these sample programs into your own applications.

The following is a list of those files that are installed in **Accusoft/ImageGear18/**

File Name	**Description**
/Bin/DL*.framework /Bin/DL*.ppi	PDF framework files to support PDF format
/Documentation/HTML/*.*	The product documentation in HTML format
/Documentation/IG_MAC.pdf	The product documentation in PDF format
/Documentation/ReleaseNotes/*.*	Release Notes of the product
/Licensing/Deployment/*.*	Server Licensing Utility for deployment purposes
/Licensing/LicenseManager/*.*	The Accusoft License Manager for evaluation and development licensing
/Samples/Xcode/ImageGearDemo/*.*	The Objective-C sample application demonstrating the use of the ImageGear Professional for Mac

By default, the ImageGear installation sets the IMAGE_GEAR_LICENSE_DIR environment variable to $HOME/Accusoft/ImageGear18/Licensing. So "accusoft.Accusoft.1-21-18.imagegear" file is placed there. However, you can change the location of the license file by changing the value of IMAGE_GEAR_LICENSE_DIR environment variable.

The ImageGear framework is installed to **Library/Frameworks/ImageGear18.framework**

1.2.2.4 Uninstalling ImageGear

To uninstall ImageGear for Mac OS X, all ImageGear installed files should be deleted.

Delete the following directories:

/Accusoft

$HOME/Accusoft/ImageGear18

/Library/Frameworks/ImageGear18.framework

To delete the ImageGear files using the user interface, run the explorer and navigate to the file or directory to delete. Call the context menu and select **Move to Trash**.

1.2.2.5 ImageGear Licensing

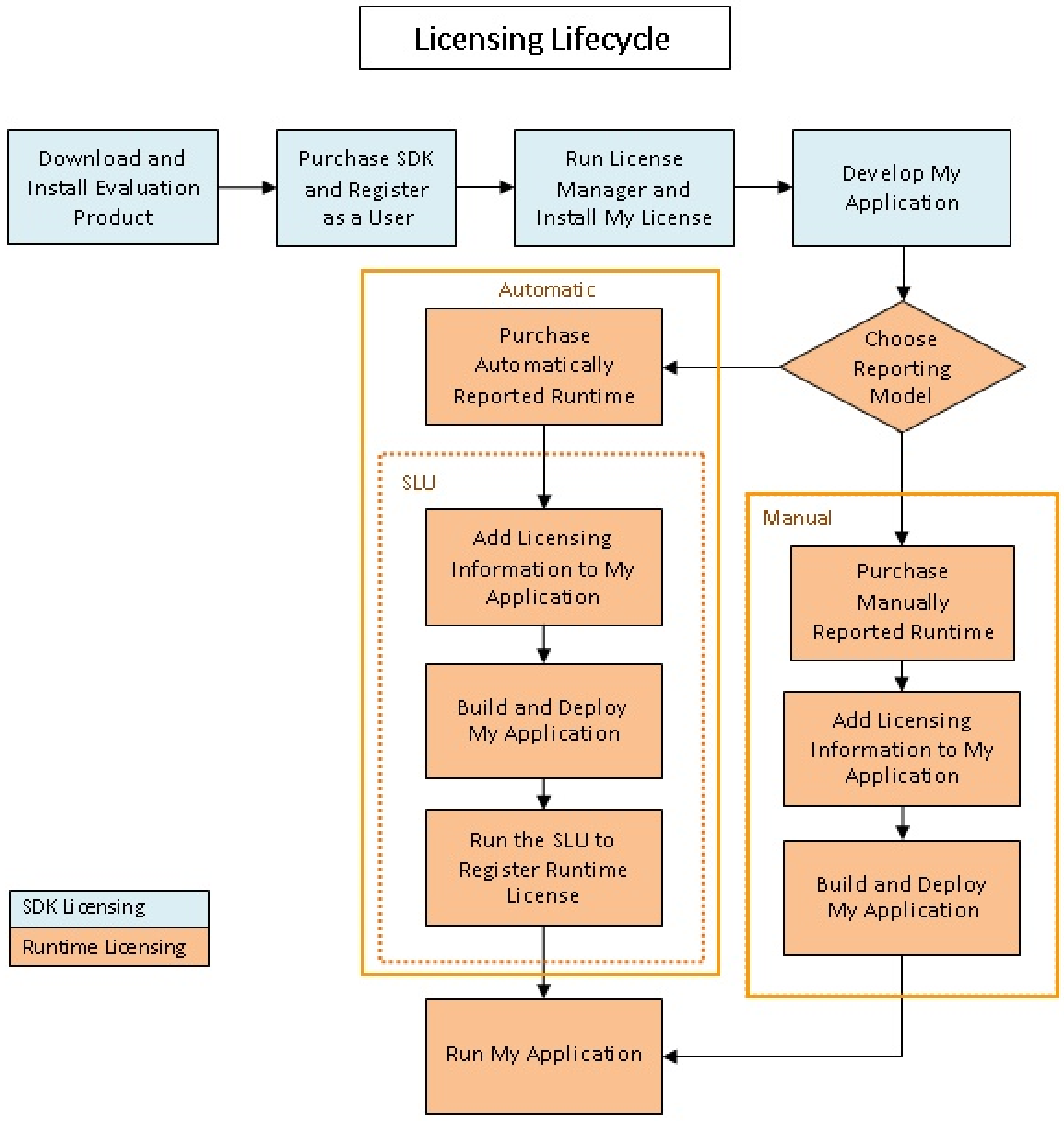
Accusoft has introduced a new licensing structure, which provides the following benefits:

- Toolkit Licensing (also known as Developer or SDK licensing) has been simplified, no longer requiring manual entry of license registration codes.
- Runtime Licensing (also known as Deployment licensing) has been made more flexible, enabling support for a number of different deployment scenarios.

During evaluation of Accusoft products, Evaluation Toolkit licenses can be used to try out products. However, the product will only function for a limited number of days since the activation of the Evaluation license.

Accusoft requires customers to purchase licenses for both development and deployment.

- Development/Toolkit Licensing: When you determine the product is a good fit, you can purchase a Toolkit license to eliminate the trial timeout while you develop your application.
- Deployment/Runtime Licensing: Once you have an application that is ready for distribution, you have options for deployment. Your own deployment scenario will dictate which option is the most appropriate. There are also cost considerations for each licensing model; see the product's "pricing" page or speak with an Accusoft Sales Representative (sales@accusoft.com) for more information.
  - Manually Reported Runtime (Non-Node-Locked) Licensing: In this model, you embed all of the licensing information directly into your application. You must manually provide royalty reporting to Accusoft for the actual licenses deployed. This model will be the best choice for you if you are not connected to the Internet at runtime, as might be the case in a defense or financial application.
  - Automatically Reported Runtime (Node-Locked) Licensing using the Server Licensing Utility (SLU): In this model, you run a small GUI tool one time on each deployment target to configure licensing. This model will be the best choice for you if you handle the deployments yourself.


Licensing Lifecycle
Download and Install Evaluation Product
Purchase SDK and Register as a User
Run License Manager and Install My License
Develop My Application
Choose Reporting Model
Automatic
Purchase Automatically Reported Runtime
SLU
Add Licensing Information to My Application
Build and Deploy My Application
Run the SLU to Register Runtime License
Manual
Purchase Manually Reported Runtime
Add Licensing Information to My Application
Build and Deploy My Application
Run My Application
SDK Licensing
Runtime Licensing

See Also:

Application Packaging

Licensing Glossary

1.2.2.5.1 License Manager

The License Manager is a GUI application that is used by a developer to register and activate Evaluation and Development (Toolkit) licenses on their development system.

Please see Evaluation Licensing or Toolkit Licensing sections for detailed instructions on using the utility for registering and activating different types of licenses.

1.2.2.5.2 Evaluation Licensing

During evaluation of Accusoft products, Evaluation Toolkit licenses can be used to try out products. However, the product will only function for a limited number of days since the activation of the Evaluation license.

No licensing calls are necessary to run the product in Evaluation mode.

> Use the License Manager to obtain an Evaluation license for your computer. Starting with ImageGear Professional v18.1, IG_lic_solution_name_set function should no longer be used to initialize Evaluation licensing.

When you determine the product is a good fit, you can purchase a Toolkit license to eliminate the trial timeout.

This section provides information about the following:

- Registering Evaluation Licenses
- Command Line Mode
- Evaluation Licensing Troubleshooting
  - Evaluation License Has Expired
  - Evaluation License Has Exceeded Installation Limit
  - Evaluation on a Device without an Internet Connection

1.2.2.5.2.1 Registering Evaluation Licenses

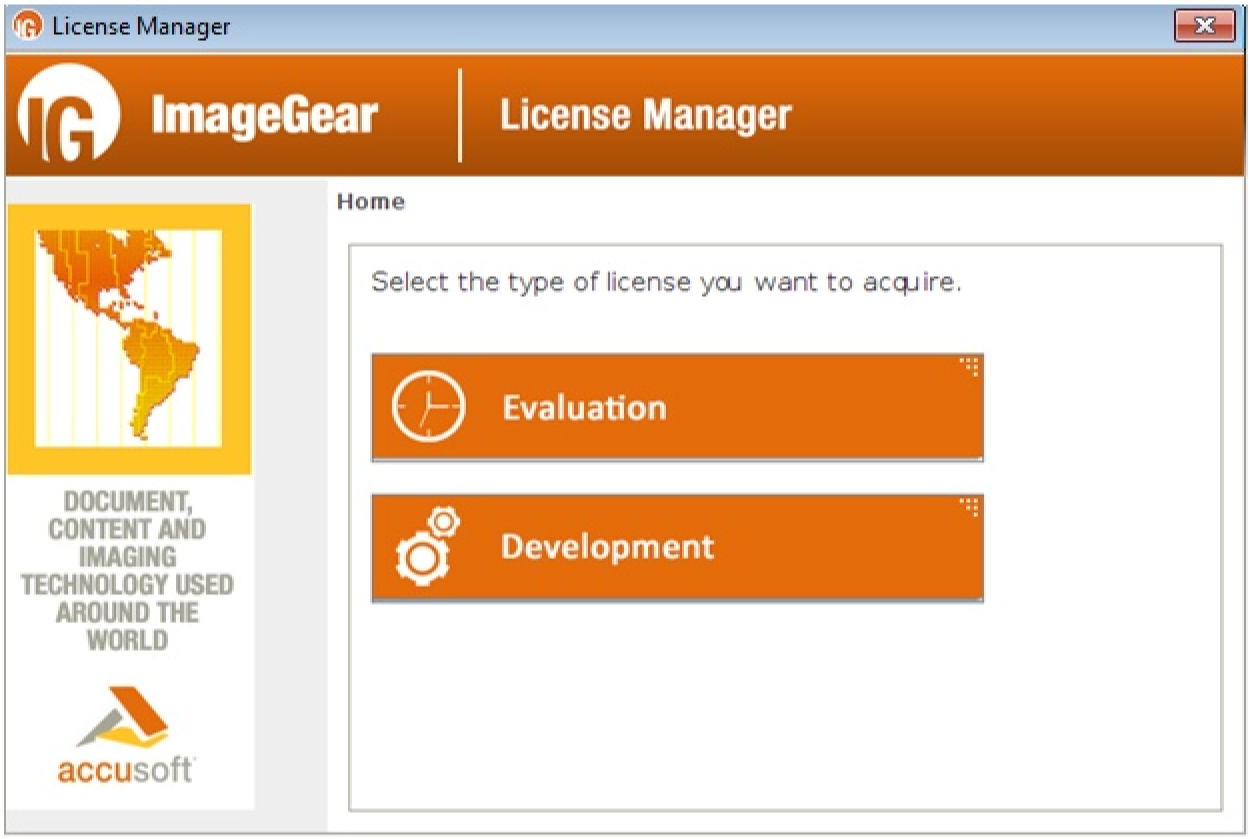
ImageGear License Manager is a GUI application that is used by a developer or end-user to register and activate Evaluation and Development (Toolkit) licenses on their system.

When an Evaluation license is registered, the License Manager:

- Communicates the developer credentials and the hardware information for the development system to the licensing web service.
- Installs the returned license key on that system.

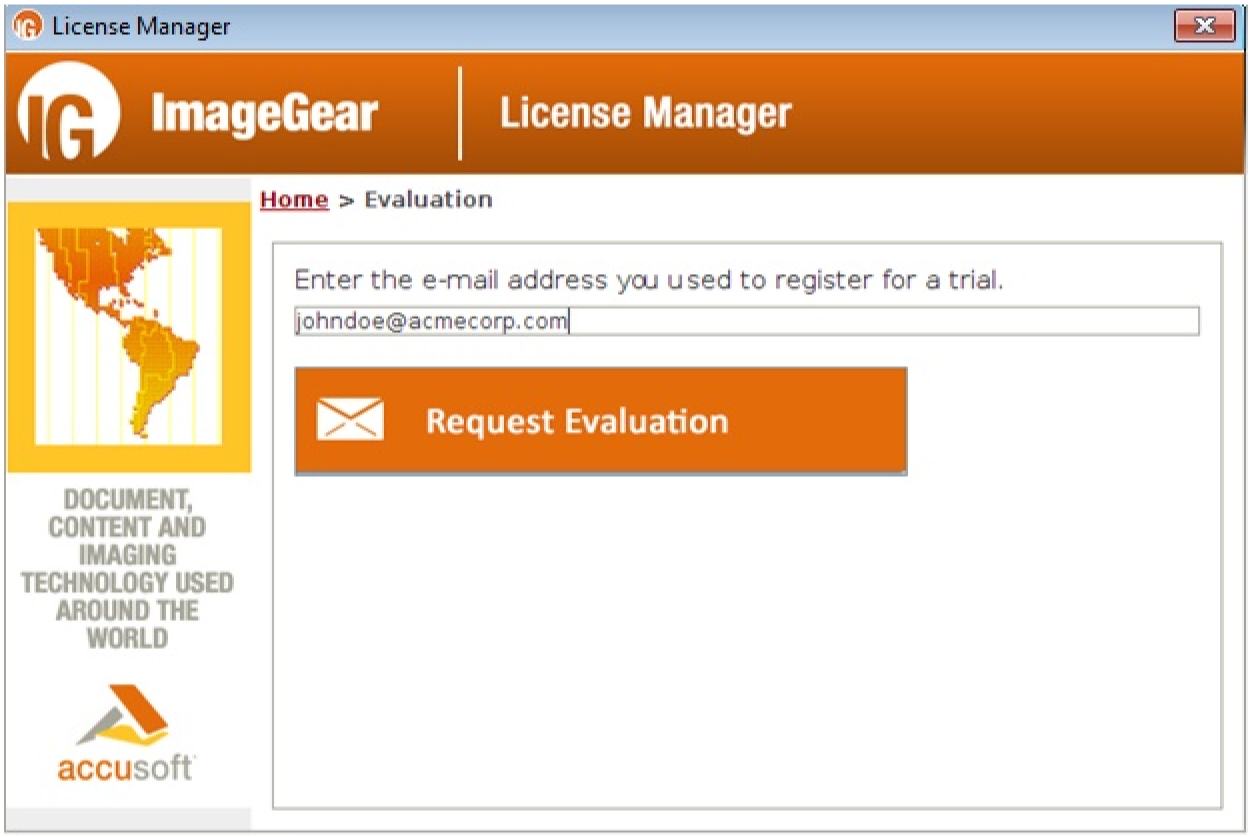
Once running, the License Manager will provide options for obtaining both Evaluation and Development licensing.

1. For Evaluation licensing, click the **Evaluation** button, as shown below.



Select the type of license you want to acquire.

2. Enter the e-mail address you used during the Evaluation registration process, and then click **Request Evaluation**.

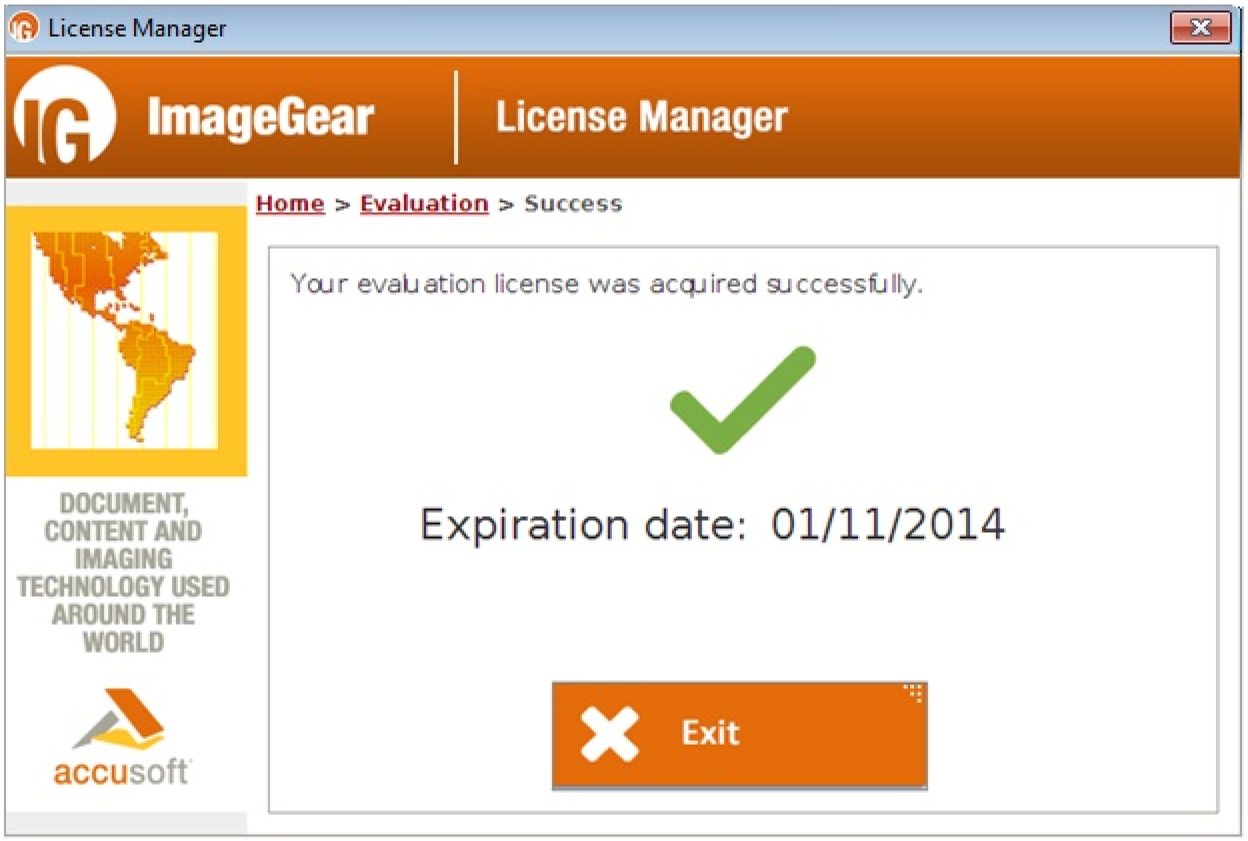

License Manager
ImageGear
License Manager
DOCUMENT, CONTENT AND IMAGING TECHNOLOGY USED AROUND THE WORLD
accusoft

Home > Evaluation

Enter the e-mail address you used to register for a trial.

johndoe@acmecorp.com

3. If your Evaluation License was acquired successfully, you should see the message below. Click **Exit** to quit the License Manager and begin your product evaluation. However, if a problem occurred during the licensing acquisition process, you may see an error. Please see Evaluation Licensing Troubleshooting for more information on resolving these potential problems.


License Manager
ImageGear
License Manager
DOCUMENT, CONTENT AND IMAGING TECHNOLOGY USED AROUND THE WORLD
accusoft

Home > Evaluation > Success

Your evaluation license was acquired successfully.

Expiration date: 01/11/2014

1.2.2.5.2.2 Command Line Mode

The License Manager can be used in command line mode for obtaining and installing evaluation licenses.

Obtaining and Installing a License from the Service

Usage:

```
eval get <e-mail> [requestextension requestinstallation outputurl]
```

Parameters:

Name	Description
<e-mail>	The e-mail address used to register for a trial. Required.
requestextension	A flag initiating a request for an evaluation extension if the evaluation license expired. Optional.
requestinstallation	A flag initiating a request for an additional installation if the limit of installations has been exceeded. Optional.
outputurl	A flag to output the URL that can be used for licensing through the web portal if there is a connectivity error. Optional.

Result Codes:

- 0 – Success
- Non-zero – Failure

Examples:

The following example demonstrates obtaining and installing an evaluation license:

```
java -jar licensemanager.jar eval get johndoe@acmecorp.com
```

The following example demonstrates obtaining and installing an evaluation license with error handling to automatically request evaluation extension, another installation, and the URL output to be used for licensing through the web portal:

```
java -jar licensemanager.jar eval get johndoe@acmecorp.com requestextension
requestinstallation outputurl
```

Installing a License Generated through the Web Portal

Usage:

```
eval write <license key>
```

Parameters:

Name	Description
<license key>	License key generated through the web portal. Required.

Result Codes:

- 0 – Success
- Non-zero – Failure

Example:

The following example demonstrates installing an evaluation license generated through the web portal:

```
java -jar licensemanager.jar eval write 2.0.YourEvaluationLicenseKey
```

1.2.2.5.2.3 Evaluation Licensing Troubleshooting

There are a few situations that may cause the request for an Evaluation License to fail. Use the table below to locate the appropriate troubleshooting topic based on the error message presented by the License Manager.

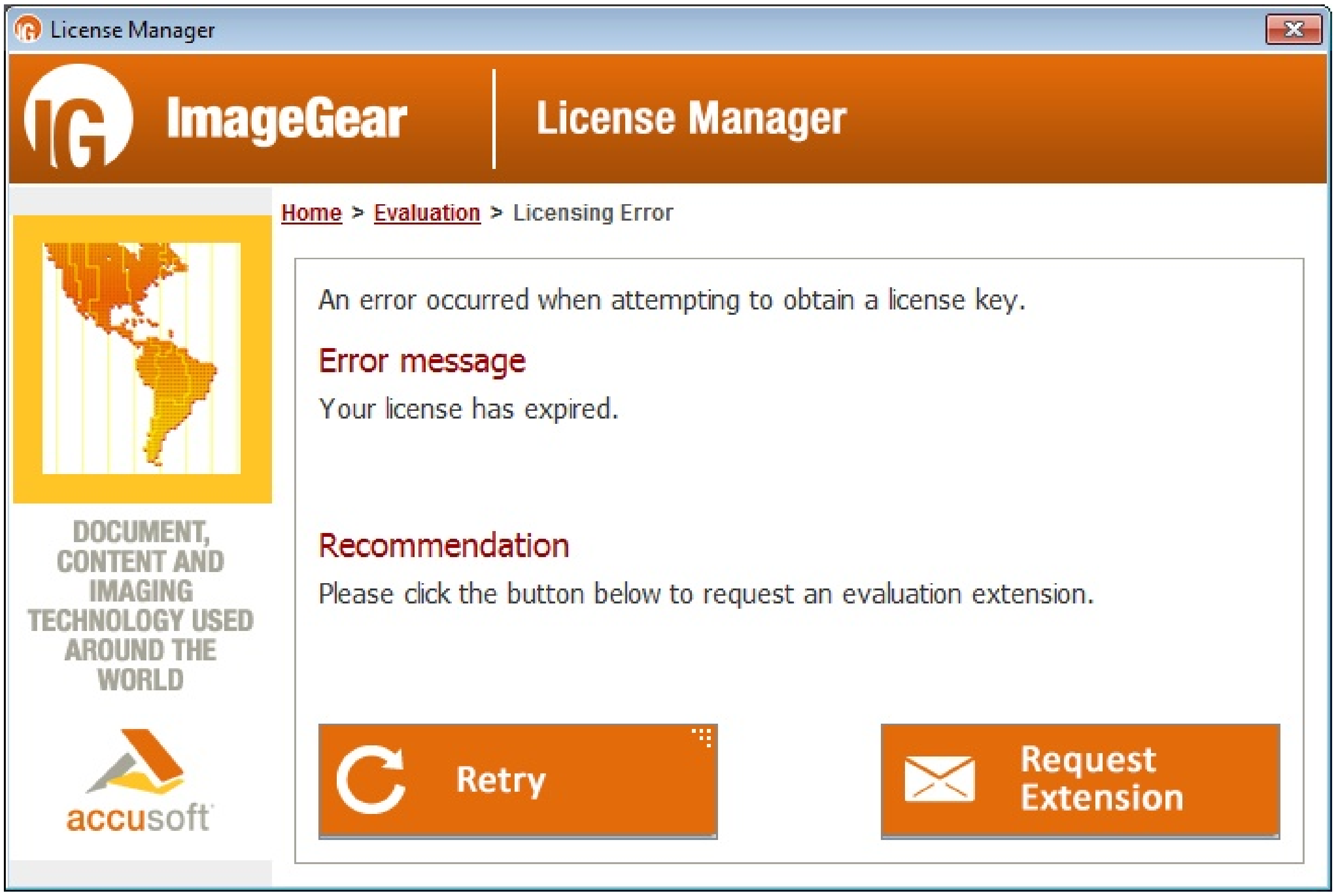
Error Message	**Resolution Topic**
Your license has expired	Evaluation License Has Expired
You have exceeded the limit of evaluation installations	Evaluation License Has Exceeded Installation Limit
Application could not reach licensing services	Evaluation on a Device without an Internet Connection

1.2.2.5.2.3.1 Evaluation License Has Expired

You may see this error returned from either the License Manager or the Evaluation Licensing website. It means that you have previously obtained an Evaluation License for your machine, using your Evaluation e-mail address, which has since expired.

To resolve this issue, you can do one of the following:

- Request an Extension - Select this option to send a request to Accusoft Sales to extend the evaluation period of your license. These requests are reviewed by Accusoft Sales staff, and are usually processed in approximately one business day.
- Purchase a Development (Toolkit) License - You may visit www.accusoft.com for pricing information and to get in contact with Accusoft Sales about purchasing a Development License.


License Manager
ImageGear
License Manager
Home > Evaluation > Licensing Error
DOCUMENT, CONTENT AND IMAGING TECHNOLOGY USED AROUND THE WORLD
accusoft
Retry
Request Extension

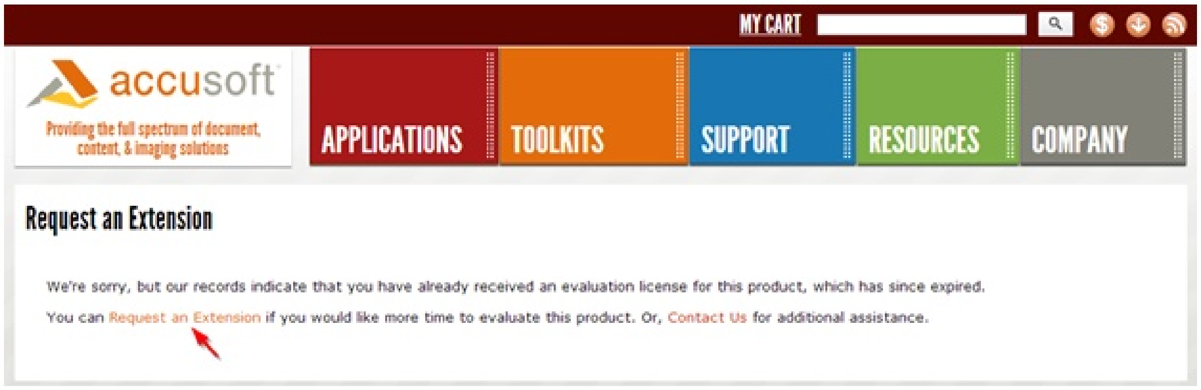
An error occurred when attempting to obtain a license key.

Error message

Your license has expired.

Recommendation

Please click the button below to request an evaluation extension.


MY CART
accusoft
Providing the full spectrum of document, content, & imaging solutions
APPLICATIONS
TOOLKITS
SUPPORT
RESOURCES
COMPANY

Request an Extension

We're sorry, but our records indicate that you have already received an evaluation license for this product, which has since expired.

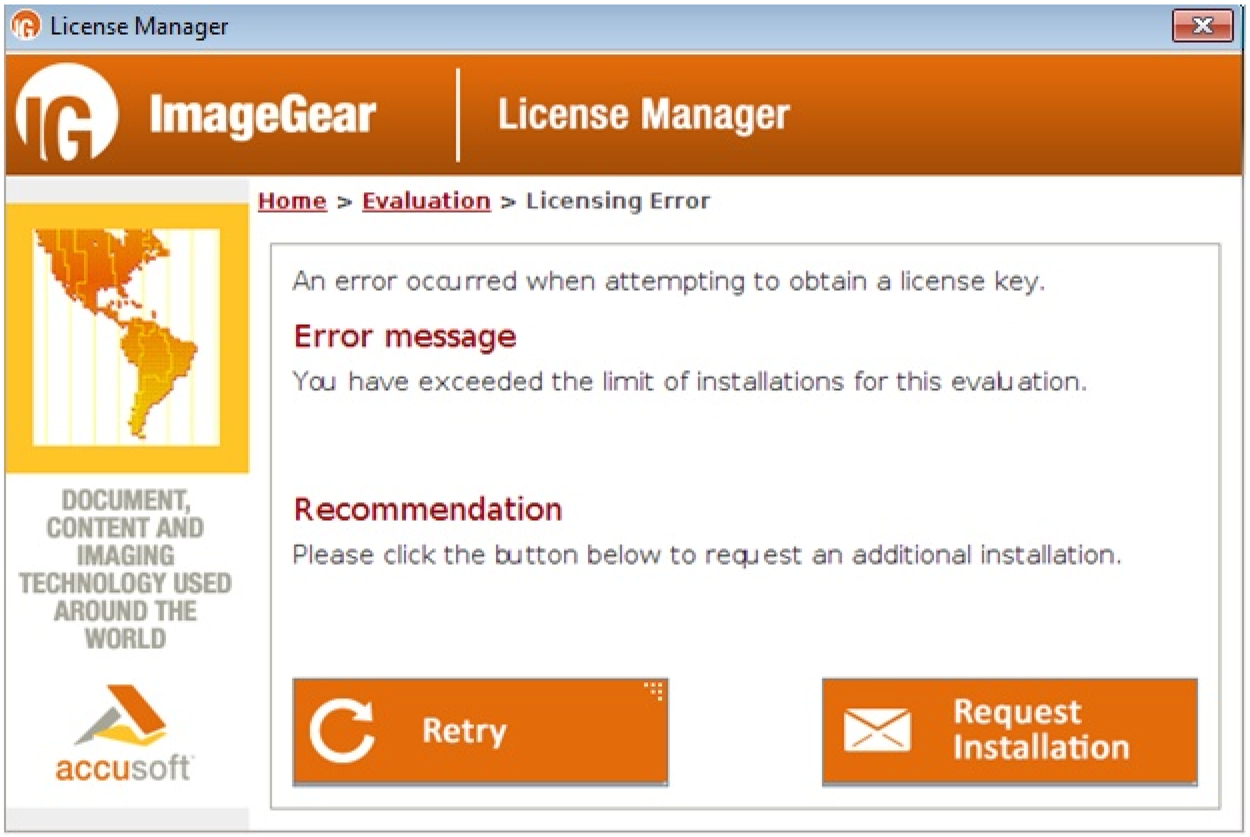
You can Request an Extension if you would like more time to evaluate this product. Or, Contact Us for additional assistance.

1.2.2.5.2.3.2 Evaluation License has Exceeded Installation Limit

You may see this error returned from either the License Manager or the Evaluation Licensing website. It means that you have obtained the maximum number of Evaluation Licenses for a specific product using your Evaluation e-mail address.

To resolve this issue, you can do one of the following:

- Request an Additional Installation - Select this option to send a request to Accusoft Sales to add an additional installation for your Evaluation License. This will allow you to obtain an Evaluation License for a new machine. These requests are reviewed by Accusoft Sales staff, and are usually processed in approximately one business day.
- Purchase a Development (Toolkit) License - You may visit www.accusoft.com for pricing information and to get in contact with Accusoft Sales about purchasing a Development License.


License Manager
ImageGear
License Manager
Home > Evaluation > Licensing Error
DOCUMENT, CONTENT AND IMAGING TECHNOLOGY USED AROUND THE WORLD
accusoft
Retry
Request Installation

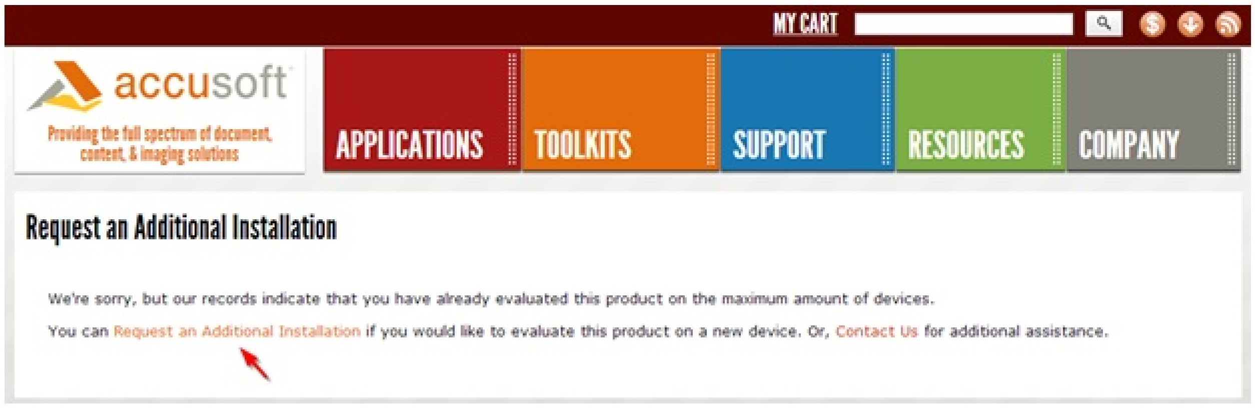
An error occurred when attempting to obtain a license key.

Error message

You have exceeded the limit of installations for this evaluation.

Recommendation

Please click the button below to request an additional installation.


MY CART
accusoft
Providing the full spectrum of document, content, & imaging solutions
APPLICATIONS
TOOLKITS
SUPPORT
RESOURCES
COMPANY

Request an Additional Installation

We're sorry, but our records indicate that you have already evaluated this product on the maximum amount of devices.

You can Request an Additional Installation if you would like to evaluate this product on a new device. Or, Contact Us for additional assistance.

1.2.2.5.2.3.3 Evaluation on a Device without an Internet Connection

Evaluation licensing can still be used on a machine without an available connection to the Internet. However, the process to acquire an Evaluation License in this situation requires a few additional steps, described below.

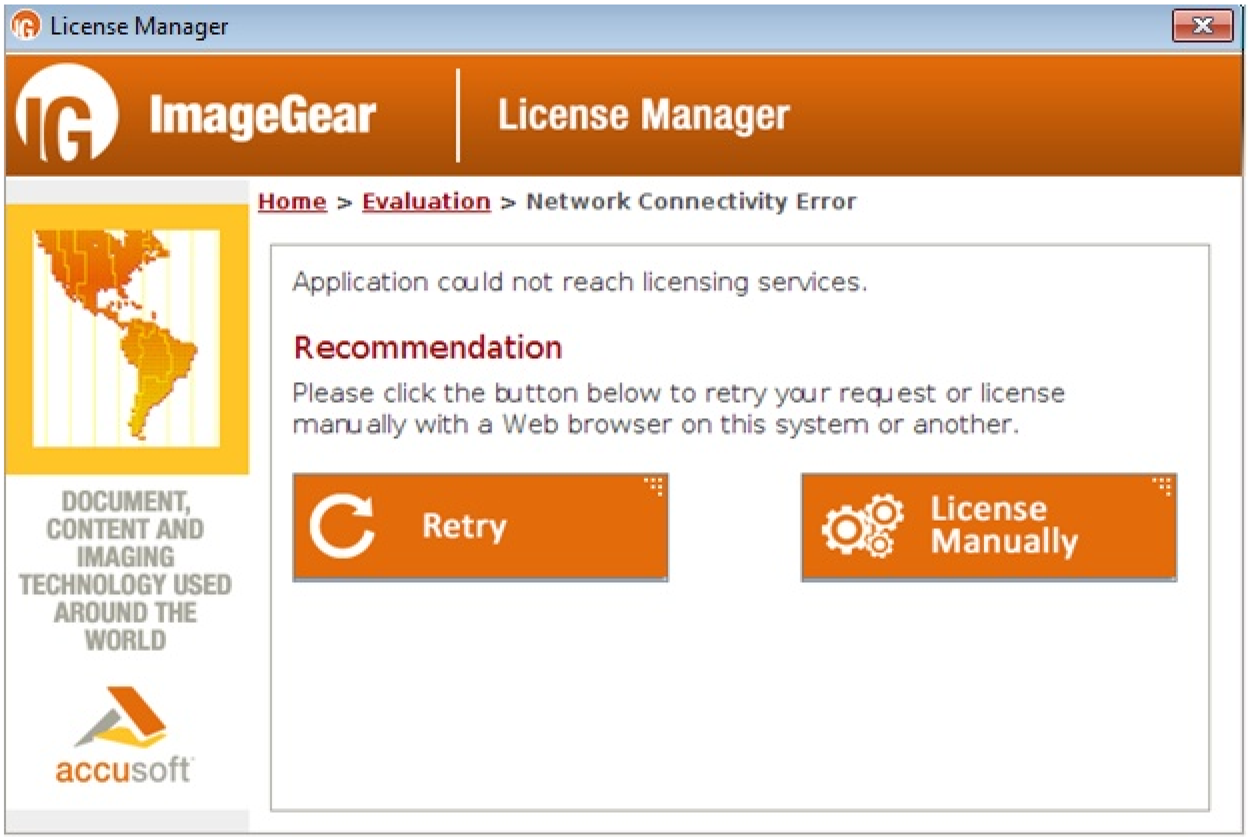
If you have not already done so, please complete the steps in Registering Evaluation Licenses before proceeding with the steps below.

1. Begin the Manual Licensing Process.

   The Manual Licensing Process may only be started once the License Manager detects that a connection to the licensing services cannot be established. The connection is attempted when **Request Evaluation** is clicked on the main Evaluation screen.

   If the machine on which you are evaluating ImageGear Professional does not have an Internet connection, you will see the screen below.

   Click **Retry** if you are aware of an Internet connection issue that has been resolved. Otherwise, click **License Manually** to begin the Manual Licensing Process.


License Manager
ImageGear
License Manager
Home > Evaluation > Network Connectivity Error
DOCUMENT, CONTENT AND IMAGING TECHNOLOGY USED AROUND THE WORLD
accusoft
Retry
License Manually

Application could not reach licensing services.

Recommendation

Please click the button below to retry your request or license manually with a Web browser on this system or another.

2. Access the Evaluation Licensing Website.

   Next, you will need to go to the Evaluation Licensing website to obtain your Evaluation license. The URL to this website is provided by the License Manager. It is important that the entire URL is used. You have two options for getting this URL:

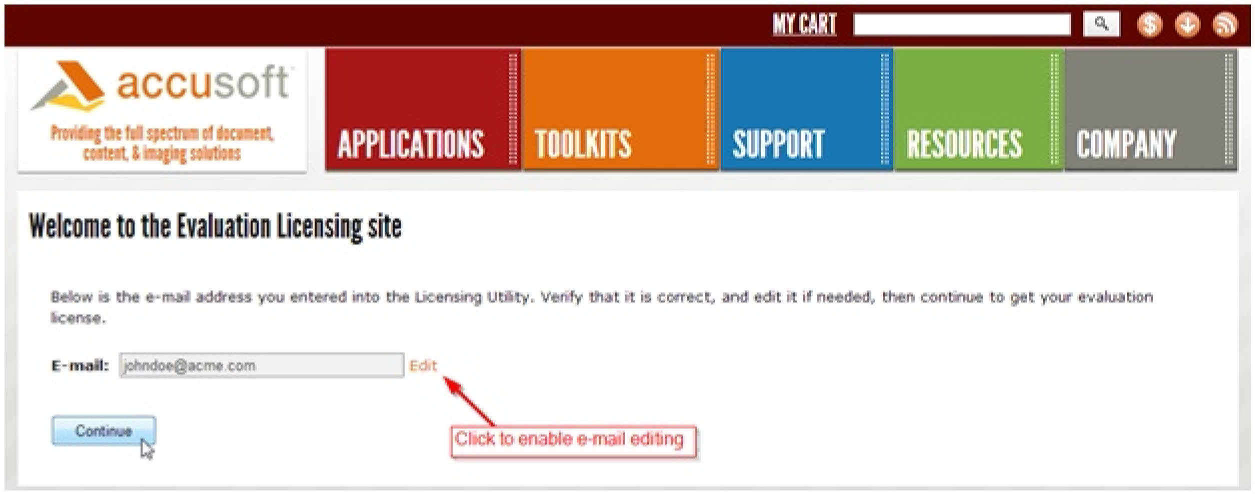
   - License on this system via Web - Choosing this option will open the default web browser on your machine and navigate directly to the Evaluation Licensing website. This option is recommended if, for example, your organization allows access to the public Internet only within the web browser through the use of proxy servers.
   - License on another system - Choosing this option will create an Internet Shortcut file (.URL). This is a simple text file that contains the full URL to the Evaluation Licensing website. In Windows environments, these files can be double-clicked to open the default web browser and navigate directly to the Evaluation Licensing website. If this action does not work in your environment, simply open the file in a text editor, copy the URL, and paste it into the address bar of your web browser. This option is recommended if the evaluation device does not have any connection to the Internet.

Click the button that is the best option in your situation to access the Evaluation Licensing website.

3. Verify Your E-mail Address and Continue.

   Once at the Evaluation Licensing website, the e-mail address you entered into the License Manager will already be pre-populated. If you notice an error in your e-mail address, you can correct it at this time by clicking the **Edit** link.

   Click the **Continue** button to obtain your Evaluation License.


MY CART
accusoft
Providing the full spectrum of document, content, & imaging solutions
APPLICATIONS
TOOLKITS
SUPPORT
RESOURCES
COMPANY
Continue
Edit
Click to enable e-mail editing

Welcome to the Evaluation Licensing site

Below is the e-mail address you entered into the Licensing Utility. Verify that it is correct, and edit it if needed, then continue to get your evaluation license.

E-mail: johndoe@acme.com

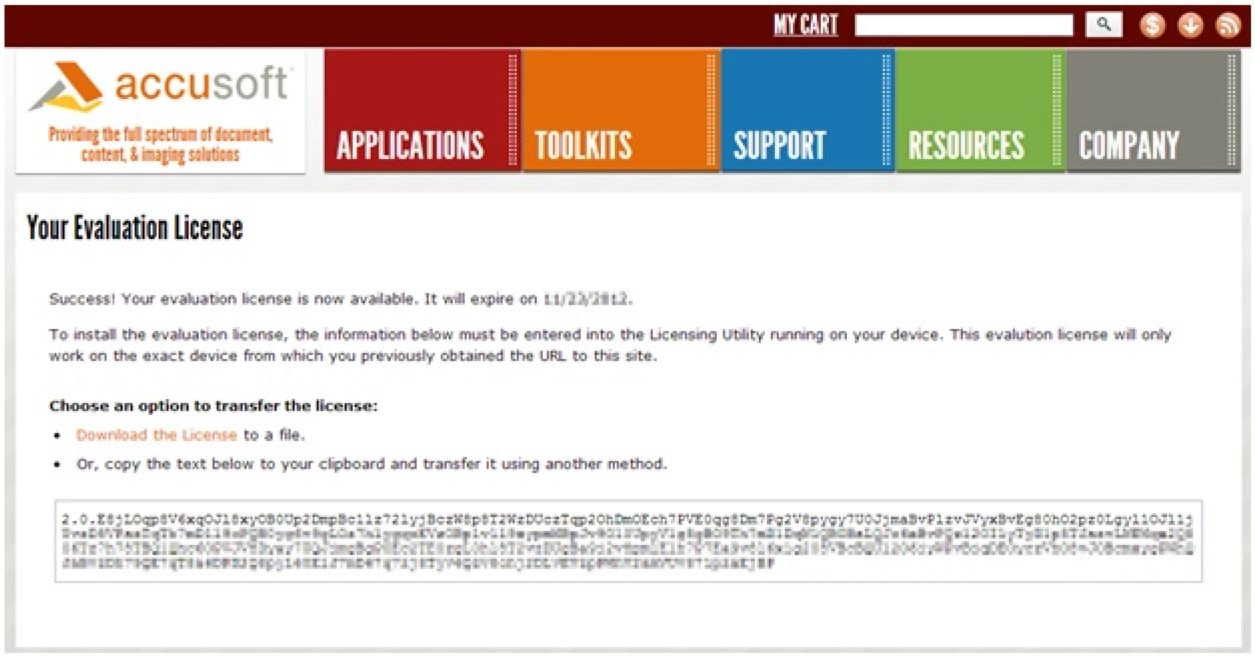
4. Transfer the License to the License Manager.

   If your Evaluation license was acquired successfully, you should see the message below. Otherwise, please see Evaluation Licensing Troubleshooting for more information on these potential problems.

   The string of alpha-numeric characters shown in the lower box is your Evaluation license. This information must be transferred back to the License Manager running on your evaluation machine.

   If you selected the option to License on this system via Web in step 3, you can simply copy the license information to your clipboard to transfer it to the License Manager.

   If you are accessing the Evaluation Licensing website from another machine, it is recommended that you download the license information to a file, and transfer the file to the evaluation machine.


MY CART
accusoft
Providing the full spectrum of document, content, & imaging solutions
APPLICATIONS
TOOLKITS
SUPPORT
RESOURCES
COMPANY

Your Evaluation License

Success! Your evaluation license is now available. It will expire on 11/23/2012.

To install the evaluation license, the information below must be entered into the Licensing Utility running on your device. This evalution license will only work on the exact device from which you previously obtained the URL to this site.

Choose an option to transfer the license:

- Download the License to a file.
- Or, copy the text below to your clipboard and transfer it using another method.

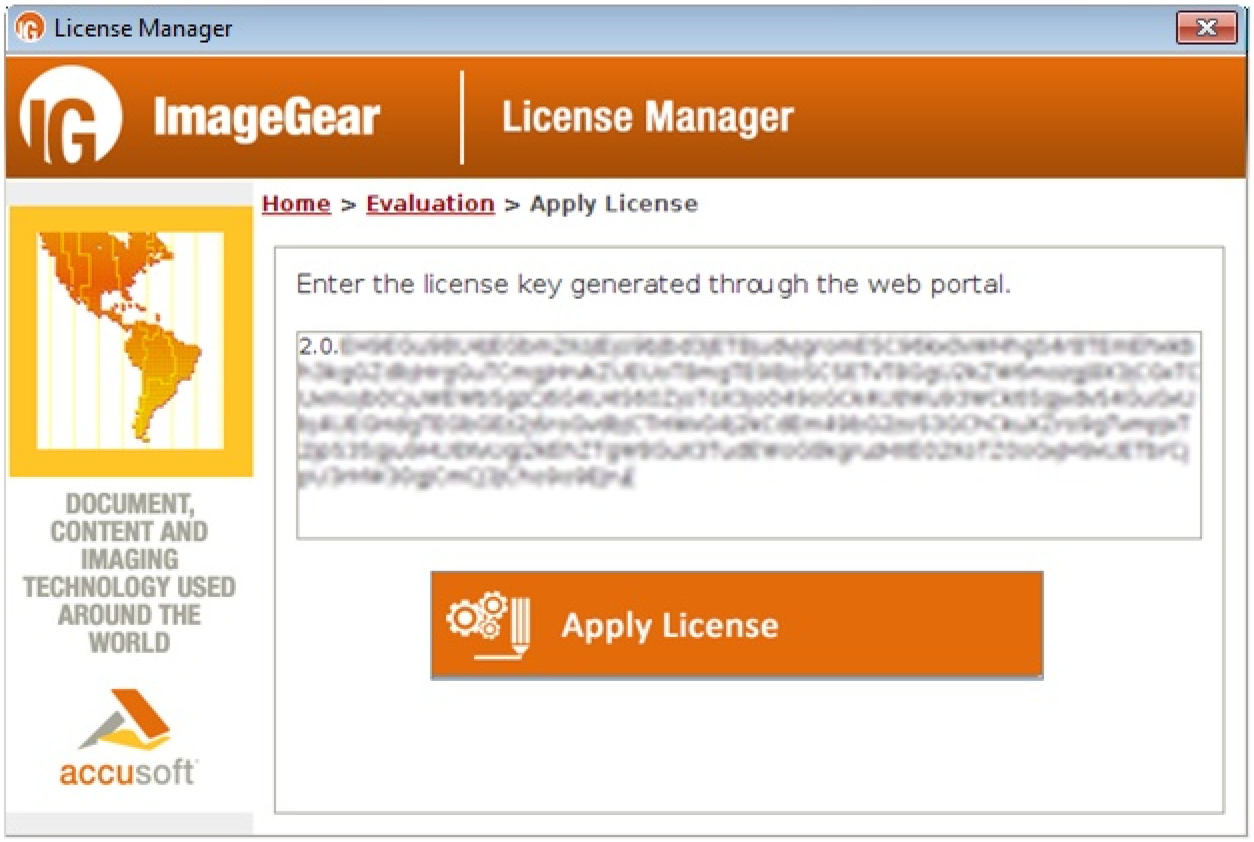
```
2.0.E8jLOgp8V6xqOJ18xyOB0Up2DmpBc11z721yjBczW8p8T2WzDUczTqp2OhDmOEch7PVE0gg8Dm7Pg2V8pygy7U0JjmaBvPlzvJVyxBvEg80hO2pz0Lgy11OJ11j
[illegible]
```

5. Enter the License into the License Manager.

   Once back to the License Manager running on your evaluation machine, you can paste the license information into the awaiting text area.

If the License Manager was closed after you left it to go to the Evaluation Licensing site, you can restart the application and perform all the previous steps again to return to this screen in the License Manager. You do not need to repeat the steps on the Evaluation Licensing website.

Enter the license information and click **Apply License** to apply the Evaluation License on the current machine.


License Manager
ImageGear
License Manager
Home > Evaluation > Apply License
DOCUMENT, CONTENT AND IMAGING TECHNOLOGY USED AROUND THE WORLD
accusoft
Apply License

Enter the license key generated through the web portal.

2.0.[redacted]

1.2.2.5.3 Toolkit Licensing

Accusoft requires customers who are developing code that uses Accusoft components to have a Toolkit license for each developer.

This section provides information about the following:

- Assigned Toolkit License
- Product Editions
- Registration
  - Registering When Connected to the Internet
  - Registering When Disconnected from the Internet
- Developing Code

1.2.2.5.3.1 Assigned Toolkit License

When licenses are purchased for developers at an organization, a representative is designated to assign the licenses to developers at the organization. Each developer must create an account on Accusoft’s website prior to the assignment of licenses. The organization’s designated representative may request that licenses be reassigned as a result of personnel changes.

1.2.2.5.3.2 Product Editions

Accusoft products may have multiple editions, each of which supports different features.

Selecting an Edition (Activating a Toolkit License)

If a customer has purchased multiple editions of the same product, they may install both editions on a development system, but only one edition may be active at a time. Once a license has been activated for a particular system, it may be deactivated (replaced by activating another edition) or reactivated at any time. Similarly, customers who own a lower featured edition may activate a full featured edition evaluation Toolkit license and then switch back to their paid, lower-featured Toolkit edition license at any time.

1.2.2.5.3.3 Registration

Registration is the process of creating a paid Toolkit License key for a particular system. Registration for Toolkit Licenses is always done through the License Manager. This removes the evaluation timeout limitation, so the toolkit will not stop functioning at the end of the evaluation period. Registration for each license is only required to be performed once per development system. After a license has been registered on a system, you can use the license without further interaction with the License Manager.

See one of the following sections for further instructions:

- Registering When Connected to the Internet
- Registering When Disconnected from the Internet

1.2.2.5.3.3.1 Registering When Connected to the Internet

When the development system is connected to the Internet, registration is a fairly simple process.

1. Start the License Manager.
2. In the list of license types to be acquired, choose **Development**.
3. When prompted, enter your Accusoft login and password. The License Manager then displays the available evaluation and purchased licenses assigned to you.
4. Select a toolkit and click the link to activate the license for that toolkit.

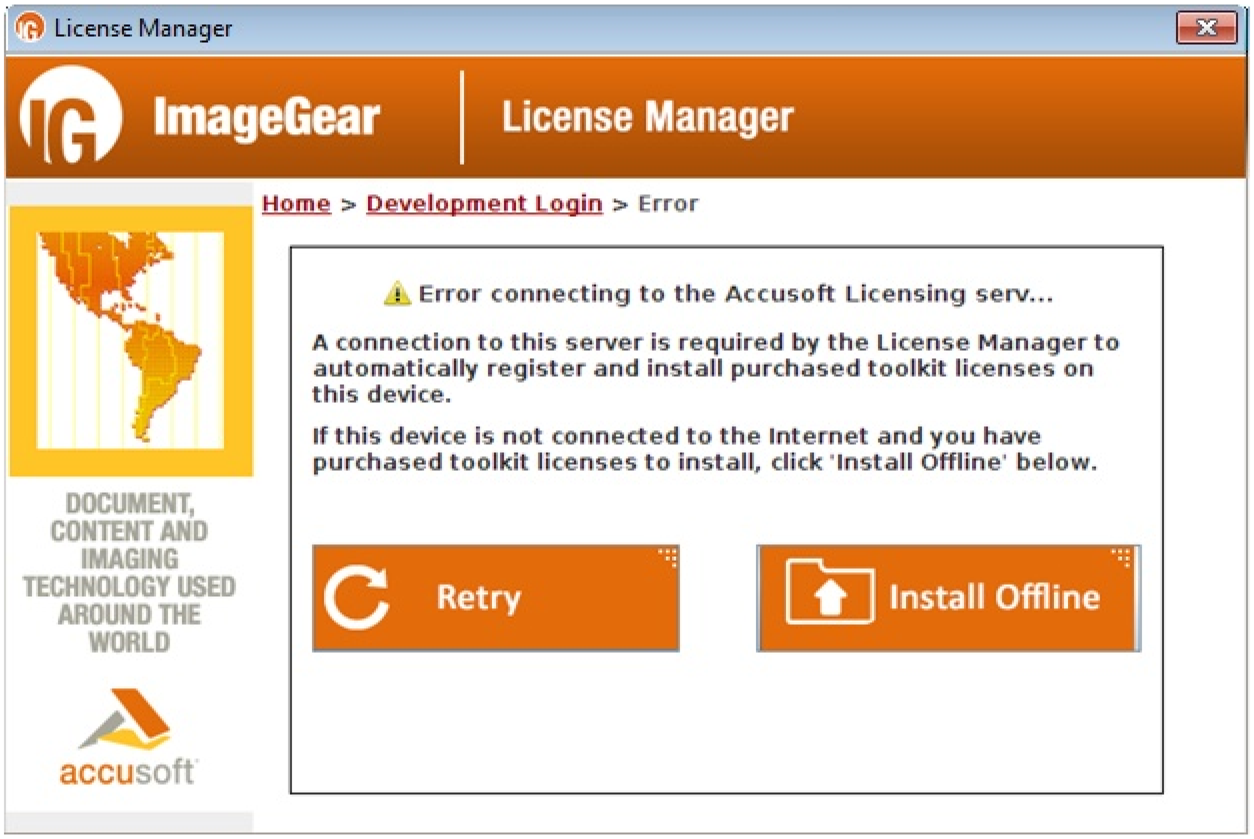
The License Manager then requests a new license key from the licensing web service and installs it; this completes the registration process.

1.2.2.5.3.3.2 Registering When Disconnected from the Internet

When the development system is not connected to the Internet during registration, as is the case in some defense or financial institutions, registration is only slightly less simple.

1. Run the License Manager and attempt to log in.

   The License Manager detects that the system is not registered and displays the following dialog:



License Manager

ImageGear | License Manager

Home > Development Login > Error

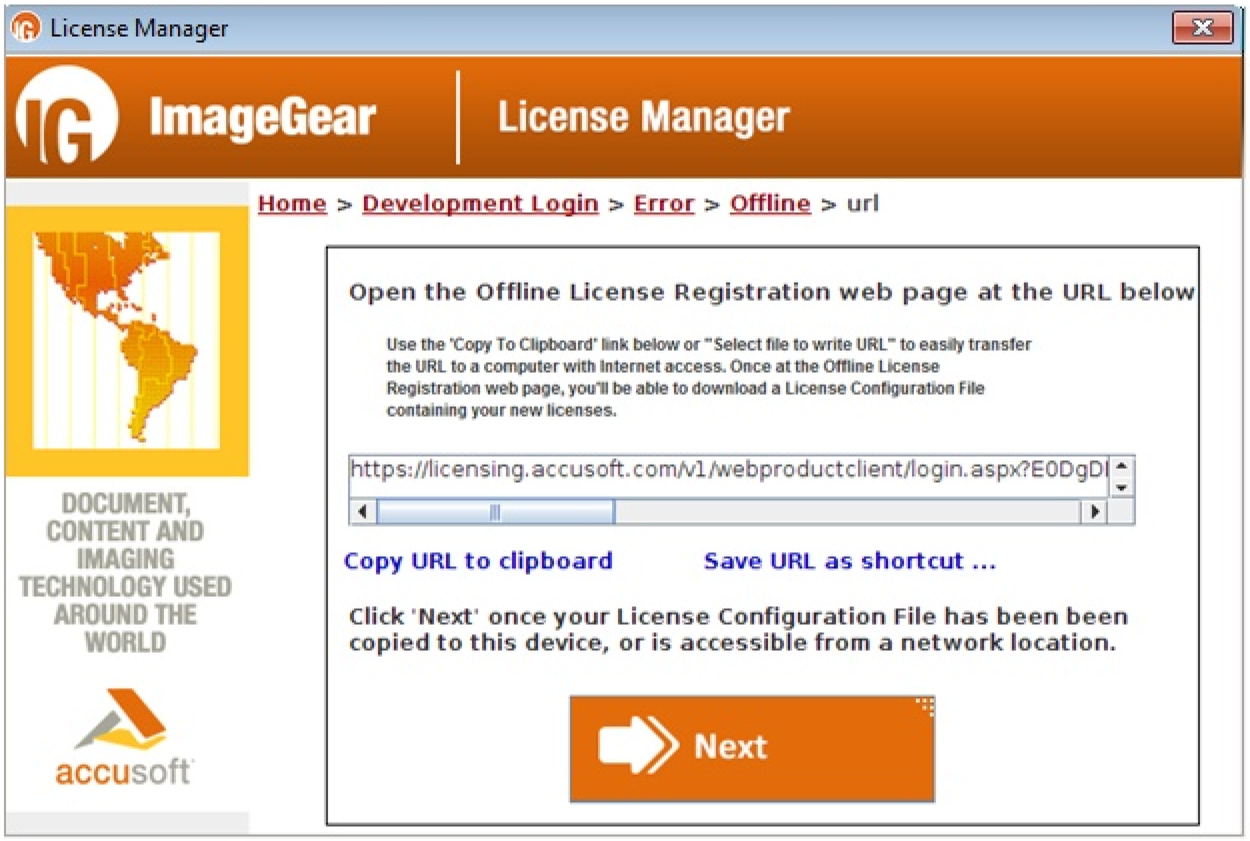
Error connecting to the Accusoft Licensing serv...

A connection to this server is required by the License Manager to automatically register and install purchased toolkit licenses on this device.

If this device is not connected to the Internet and you have purchased toolkit licenses to install, click 'Install Offline' below.

Retry
Install Offline
DOCUMENT, CONTENT AND IMAGING TECHNOLOGY USED AROUND THE WORLD
accusoft

2. Select the Install Offline option to begin the offline registration process.


License Manager
ImageGear
License Manager
Home > Development Login > Error > Offline > url
DOCUMENT, CONTENT AND IMAGING TECHNOLOGY USED AROUND THE WORLD
accusoft
Copy URL to clipboard
Save URL as shortcut ...
Next

Open the Offline License Registration web page at the URL below

Use the 'Copy To Clipboard' link below or "Select file to write URL" to easily transfer the URL to a computer with Internet access. Once at the Offline License Registration web page, you'll be able to download a License Configuration File containing your new licenses.

https://licensing.accusoft.com/v1/webproductclient/login.aspx?E0DgD

Click 'Next' once your License Configuration File has been been copied to this device, or is accessible from a network location.

3. Copy the URL to removable media, such as a thumb drive.
4. Take the thumb drive to a system that is connected to the Internet.
5. From the connected system, paste the URL into a browser, which automatically displays the toolkits available.
6. Upon selecting a toolkit, a license is generated, which you then save to the removable media device.
7. Return to the offline system where the new license is to be installed.
8. Use the License Manager to browse to the file and install the Toolkit license.

1.2.2.5.3.4 Developing Code

Once an evaluation or paid Toolkit license has been activated on the development system, no additional code is required to use the Accusoft products on this system. If moving the resulting executable code to a new system, you will need to install Toolkit licenses or incorporate Runtime licenses into your code.

1.2.2.5.4 Runtime Licensing

When an application that uses one or more Accusoft products is deployed, it requires a Runtime License for each installation. There are two main types of Runtime Licenses:

- Automatically Reported Runtime (Node-Locked). Distribution of Automatically Reported Runtime licenses is handled automatically through the use of licensing software components and web services.
- Manually Reported Runtime (Non-Node-Locked). Manually Reported Runtime licensing requires you to provide the number of licenses distributed on a contractually agreed upon basis.

These licensing types are each described in detail in the following sections:

- Automatically Reported Runtime (Node-Locked)
    - Licensing API
    - License Pools
    - License Configuration Files
    - Server Licensing Utility (SLU)
        - Command Line Mode
- Manually Reported Runtime (Non-Node-Locked)

1.2.2.5.4.1 Automatically Reported Runtime (Node-Locked)

Automatically Reported Runtime licenses use a mechanism similar to Toolkit licensing to activate a license for a particular system. In this case, each installation has a unique license that is pulled from the pool of licenses purchased by the customer. When the license key is generated, it contains hardware information that identifies the system on which the license is to be installed.

The Server Licensing Utility (SLU) is a stand-alone application that interacts with the licensing web service and installs the newly generated license key to perform the registration.

This topic provides information about the following:

- Licensing API
- License Pools
- License Configuration Files
- Server Licensing Utility (SLU)
    - Command Line Mode

1.2.2.5.4.1.1 Licensing API

The Licensing API is used in the developer's code to specify Runtime Licensing deployment information. This is used to unlock the Accusoft products and enable all licensed features at runtime. Both the "Solution Name" and "Solution Key" values used in the API described below are provided by Accusoft, along with the License Configuration file, at the time of purchase of a Runtime License.

Solution Name

The solution name is the name assigned by Accusoft to the licenses purchased for runtime deployment. It is a character string that is set for a component prior to use in a deployment environment and is typically the name of the organization that purchased the runtime licenses. It is set via the IG_lic_solution_name_set function. For example:

```
IG_lic_solution_name_set("ACMEImaging");
```

The Solution Name must be set in order to use runtime licensing.

Solution Key

The Solution Key is a set of four numbers assigned when the licenses are purchased for runtime deployment; the Solution Key also identifies the organization that purchased the runtime licenses. The Solution Key is set via the IG_lic_solution_key_set function. For example:

```
IG_lic_solution_name_set("ACMEImaging");
IG_lic_solution_key_set(0x1C3A023F, 0xA018F260, 0x37AF0E51, 0x557F2389);
```

The Solution Key must be set in order to use runtime licensing.

1.2.2.5.4.1.2 License Pools

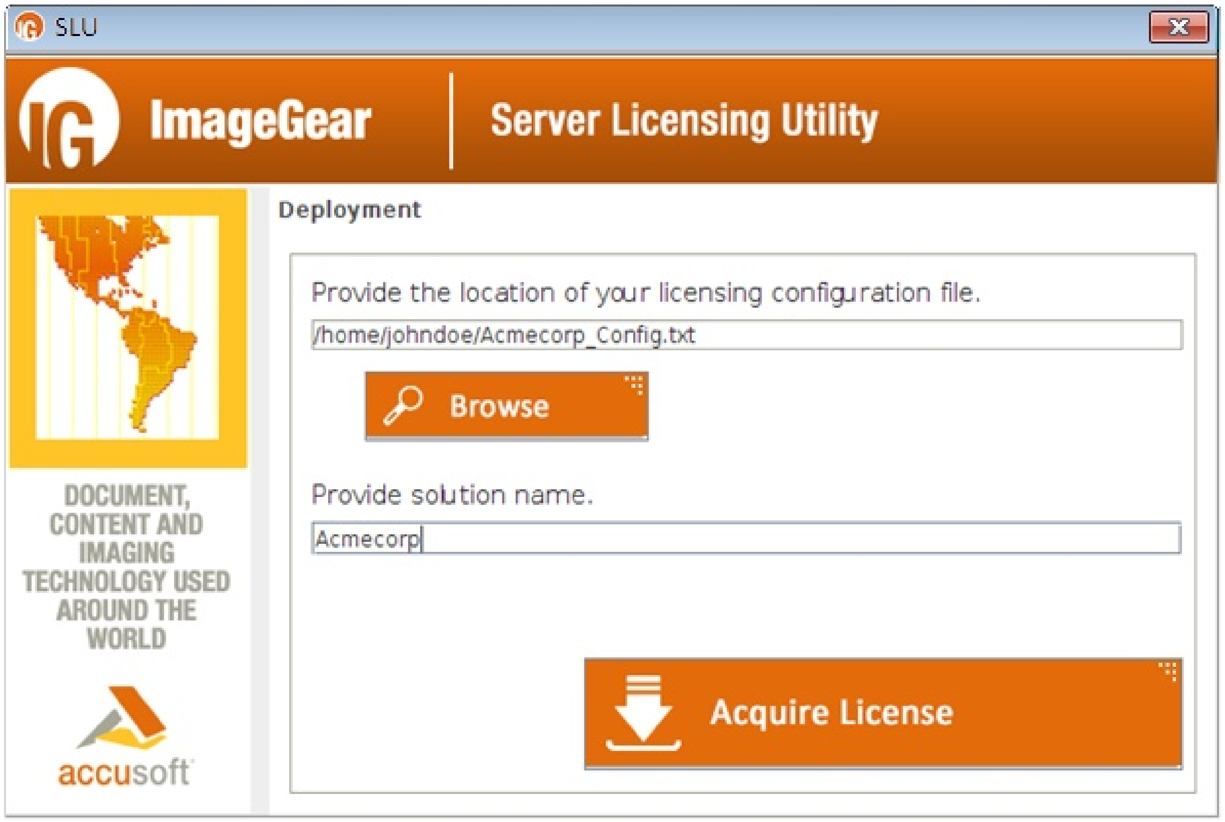
When Automatically Reported Runtime licenses are purchased, they are organized into pools of a specific version and platform for a particular toolkit. For example, if a customer developed two applications using ImageGear, one using 32-bit DLL and the other using .NET, they would purchase two pools of runtime licenses, one for the DLL platform and one for the .NET platform. At installation, a new license would be pulled from the appropriate pool, depending upon which platform the application is using.

1.2.2.5.4.1.3 License Configuration Files

When an Automatically Reported Runtime license is purchased, a License Configuration file is created and distributed to the customer along with their assigned Solution Name and Solution Key. The License Configuration file contains information about the Runtime licenses for a particular platform and version of a toolkit. It is used by the Server Licensing Utility to activate licenses when the customer’s application is installed on their users' systems.

1.2.2.5.4.1.4 Server Licensing Utility (SLU)

The Server Licensing Utility (SLU) uses License Configuration Files to request a license key from the purchased Runtime licenses. The SLU registers that Runtime license on the system where your application will be running. This utility can be used by developers for testing, by your deployment team, or by your end users to register their Runtime license.



Provide the location of your licensing configuration file.

/home/johndoe/Acmecorp_Config.txt

Provide solution name.

Acmecorp

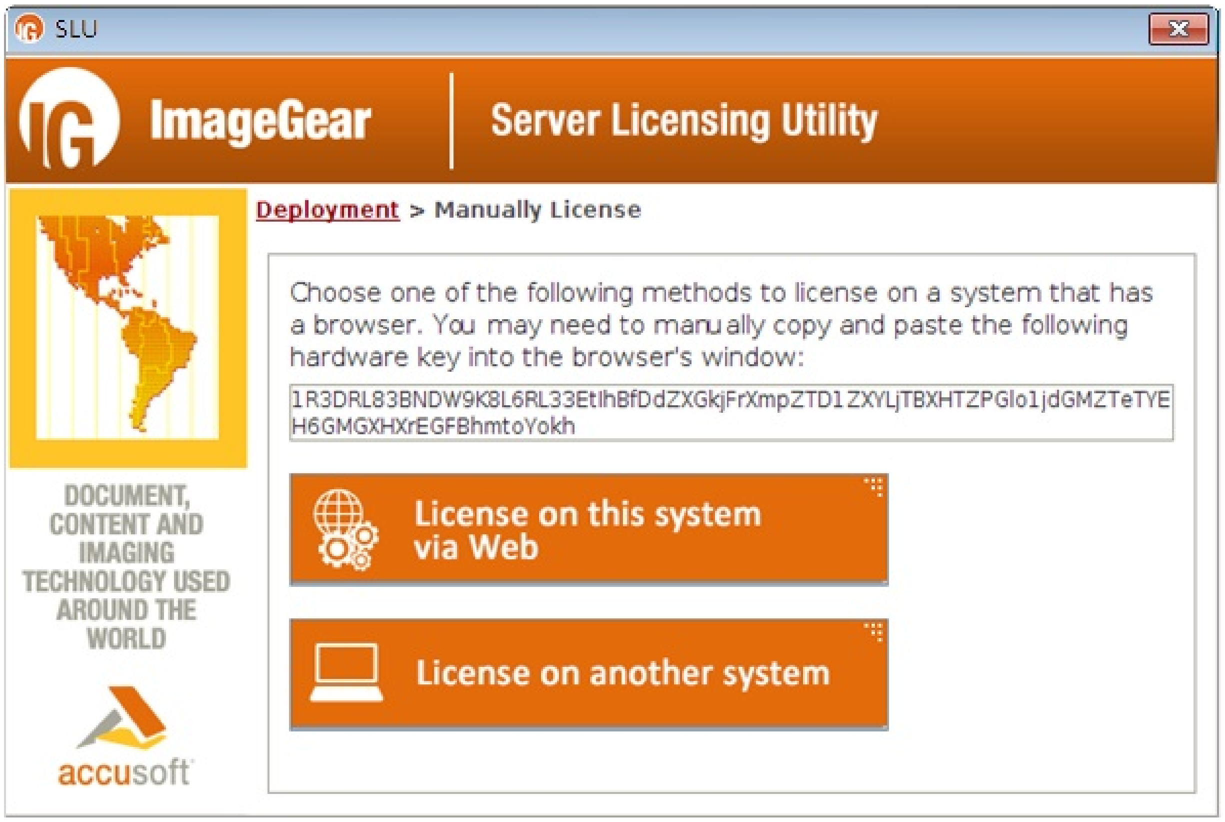
Use the following steps to register the license on your system:

1. Run the SLU using **runSlu** shell script.
2. Provide the location of the license configuration file (type in the corresponding text box or use the **Browse** button) and the solution name.
3. Click **Acquire License**.
   - Automatic Registration (Connected to Internet)

     Automatic registration works in much the same way as connected Toolkit registration and requires an Internet connection on the system where your application software will be deployed. The license information along with the system's hardware information is sent over the Internet to the Accusoft licensing web service. If an unused license is available, a new license key containing information for the system is generated, returned, and is then automatically installed by the Server Licensing Utility.
   - Manual Registration (Disconnected from Internet)

     In the situation where the SLU is not able to contact the Licensing services, a dialog will be displayed stating that the "application could not reach the licensing services". You will have the option to retry the registration or to "License Manually". Select the **License Manually** option to proceed.

     a. The Manually License dialog will display a text box with your system Hardware Key. This key is used to identify your system during the registration process. This key will need to be supplied to the Accusoft Licensing Center in order to obtain a license to register the system. Using your mouse or keyboard select all of the text within the text box and copy it to the clipboard.


SLU
ImageGear
Server Licensing Utility
DOCUMENT, CONTENT AND IMAGING TECHNOLOGY USED AROUND THE WORLD
accusoft
License on this system via Web
License on another system

Deployment > Manually License

Choose one of the following methods to license on a system that has a browser. You may need to manually copy and paste the following hardware key into the browser's window:

1R3DRL83BNDW9K8L6RL33EtlhBfDdZXGkjFrXmpZTD1ZXYLjTBXHTZPGlo1jdGMZTeTYEH6GMGXHXrEGFBhmtoYokh

b. Next, you will need to go to the Accusoft Licensing Center to obtain your Evaluation license. The URL to this website is provided by the SLU, https://licensing.accusoft.com/v1/WebDeployUser/WebDeployUser.aspx. You have two options for getting this URL:
   - License on this system via Web - Choosing this option will open the default web browser on your machine and navigate directly to the Accusoft Licensing Center. This option is recommended if, for example, your organization allows access to the public Internet only within the web browser through the use of proxy servers.
   - License on another system - Choosing this option will create an Internet Shortcut file (.URL). This is a simple text file that contains the full URL to the Accusoft Licensing Center website. These files can be double-clicked to open the default web browser and navigate directly to the Accusoft Licensing Center website. If this action does not work in your environment, simply open the file in a text editor, copy the URL, and paste it into the address bar of your web browser. This option is recommended if the System being registered does not have any connection to the Internet.

c. Once you have navigated your web browser to the Accusoft Licensing Center, you will need to enter your Hardware Key into the text box labeled Hardware Key.

d. Click the **Download License** button to have the Accusoft Licensing Center generate a license for your system. The License will be created and sent to your system as a text file.

e. Enter the License into the Server Licensing Utility.

Once back to the SLU running on your system, you can paste the license information into the awaiting text area.

If the SLU was closed after you left it to go to the Accusoft Licensing Center, you can restart the application and perform all the previous steps again to return to this screen in the Server Licensing Utility. You do not need to repeat the steps on the Accusoft Licensing Center web site.

Enter the license information and click **Apply License** to apply the License on the current machine.

1.2.2.5.4.1.4.1 Command Line Mode

The Server Licensing Utility can be used in command line mode for obtaining and installing Runtime Licenses.

Obtaining and Installing a License from the Service

Usage:

```
deploy get <configuration file><solution name> [<access key>outputurl]
```

Parameters:

Name	Description
<configuration file>	Path to the license configuration file. Required.
<solution name>	Solution name for Runtime Licensing. Required.
<access key>	Access key for annual Runtime Licensing. Optional.
outputurl	A flag to output the URL and system Hardware Key that can be used for licensing through the web portal if there is a connectivity error. Optional.

Result Codes:

- 0 – Success
- Non-zero – Failure

Examples:

The following example demonstrates obtaining and installing Runtime License:

```
java -jar slu.jar deploy get "/Path to/YourSolutionName_Config.txt" "YourSolutionName"
```

The following example demonstrates obtaining and installing Runtime License for the provided access key:

```
java -jar slu.jar deploy get "/Path to/YourSolutionName_Config.txt" "YourSolutionName"
Your-Access-Key
```

The following example demonstrates obtaining and installing Runtime License with error handling to automatically output URL and system Hardware Key to be used for licensing through the web portal:

```
java -jar slu.jar deploy get "/Path to/YourSolutionName_Config.txt" "YourSolutionName"
outputurl
```

Installing a License Generated through the Web Portal

Usage:

```
deploy write <solution name><license key>
```

Parameters:

Name	Description
<solution name>	Solution name for Runtime Licensing. Required.
<license key>	License key generated through the web portal. Required.

Result Codes:

- 0 – Success
- Non-zero – Failure

Example:

The following example demonstrates installing a Runtime License generated through the web portal:

```
java -jar slu.jar deploy write "YourSolutionName"2.0.YourDeploymentLicenseKey
```

1.2.2.5.4.2 Manually Reported Runtime (Non-Node-Locked)

Manually Reported Runtime licensing embeds all of the licensing information directly into your application. Installation does not require any further licensing interaction. However, it is your responsibility to provide royalty reporting to Accusoft for the actual licenses deployed.

This section provides information about the following:

- Solution Name
- Solution Key
- OEM License Key

Solution Name

The solution name is

- A name assigned by Accusoft to the licenses purchased for runtime deployment.
- A character string that is set for a component prior to use in a deployment environment.
- Typically the name of the organization that purchased the runtime licenses.
- Set via the IG_lic_solution_name_set function. For example:

  C++

  ```
  IG_lic_solution_name_set("ACMEImaging");
  ```

The Solution Name must be set in order to use runtime licensing.

Solution Key

The Solution Key is

- A set of four numbers assigned when the licenses are purchased for runtime deployment.
- An identifier of the organization that purchased the runtime licenses.
- Set via the IG_lic_solution_key_set function. For example:

  C++

  ```
  IG_lic_solution_name_set("ACMEImaging");
  IG_lic_solution_key_set(0x1C3A023F, 0xA018F260, 0x37AF0E51, 0x557F2389);
  ```

The Solution Key must be set in order to use runtime licensing.

OEM License Key

The OEMLicense Key is

- For use only by customers who choose to use Manually Reported Runtime licensing.
- A unique identifier of the customer, product, version, edition, and platforms for which the license is valid.
- When this key is set via the IG_lic_OEM_license_key_set function, the component is completely licensed and can be distributed without any further licensing operations.

  C++

  ```
  IG_lic_solution_name_set("ACMEImaging");
  IG_lic_solution_key_set(0x1C3A023F, 0xA018F260, 0x37AF0E51, 0x557F2389);
  IG_lic_OEM_license_key_set("2.0.GQCC0EmUgONaI4QDZ32tpGWfpGW4gtbC0iIC...");
  ```

This method is not required for Automatically Reported Runtime licensing.

1.2.2.5.5 Application Packaging

Regardless of your deployment model, the first step in deploying your application is to package the required ImageGear runtime components.

The following content of the /Accusoft/ImageGear18/Bin/.. from your ImageGear Installation represents the APDFL framework components for working with PDF files:

- DL*.framework.
- ICU*.framework.

These frameworks should be copied at the "Copy Files" build phase to Frameworks destination (see ImageGearDemo sample for reference).

Also, ImageGear18.framework must be installed to /Library/Frameworks directory on the end-user machine.

If you are using Automatically Reported Runtime (Node-Locked) Licensing model:

- If you are using the Server Licensing Utility (SLU) to register licenses on the end user's system, the "slu.jar" and "ldk.jar" located in /Accusoft/ImageGear18/Licensing/Deployment/ have to be distributed.
- Also please make sure your installation routine defines the IMAGE_GEAR_LICENSE_DIR environment variable that contains a path to the "accusoft...imagegear" license file. The end user will obtain the license file and place it to the directory indicated by IMAGE_GEAR_LICENSE_DIR environment variable.

Example of a script to set IMAGE_GEAR_LICENSE_DIR environment variable:

```
#!/bin/sh
mkdir -p $HOME/Accusoft/ImageGear18/Licensing
# Set environment variable permanently
if ! -s "/etc/launchd.conf" ; then
  touch /etc/launchd.conf
fi
if ! grep -q 'IMAGE_GEAR_LICENSE_DIR' "/etc/launchd.conf" ; then
  echo "Editing /etc/launchd.conf to add IMAGE_GEAR_LICENSE_DIR"
  echo "setenv IMAGE_GEAR_LICENSE_DIR $HOME/Accusoft/ImageGear18/Licensing" >>
"/etc/launchd.conf"
fi
echo "Temporary set IMAGE_GEAR_LICENSE_DIR" for current session
launchctl setenv IMAGE_GEAR_LICENSE_DIR $HOME/Accusoft/ImageGear18/Licensing
export IMAGE_GEAR_LICENSE_DIR=$HOME/Accusoft/ImageGear18/Licensing
```

1.2.2.5.6 Licensing Glossary

The following sections each describe a licensing term:

Access Key

All licenses are assigned a unique identifier known as an Access Key. Access Keys are associated with the organization that purchased the license.

Accusoft Products

An Accusoft product may be licensed with either a Toolkit License or a Runtime License.

Activation

Activation is the process of selecting a previously registered Toolkit license. Licenses may be paid (Toolkit) or evaluation, and may be for one of many product editions, for products with multiple editions.

Automatically Reported Runtime Licensing

Runtimes may be licensed in one of two ways: automatic reporting or manual reporting. With automatic reporting, you do not need to worry about royalty reporting; it is handled by the licensing layer. See Automatically Reported Runtime (Node-Locked).

Customers, Developers, and Users

Customers are simply any person who has purchased an Accusoft product. Developers are customers who possess a Toolkit license. Users are typically the customers of Accusoft's customers who use applications built around Accusoft components and are assigned a runtime license.

Edition

Some Accusoft products have multiple editions. Editions may offer multiple levels of product speed or product features, allowing customers to find a price-performance mix that is appropriate for them. Products that support multiple editions register multiple Toolkit licenses at installation time, one Toolkit license per edition. Developers must select an edition to activate using the License Manager.

Evaluation License

An evaluation license is a Toolkit license that is unpaid. When an Accusoft Toolkit is installed, evaluation licenses for all applicable product editions are installed and registered, and the edition with the most features is activated. Evaluation Toolkit licenses can be used to try out products. However, the product behavior is limited by trial dialog pop-ups. When you determine the product is a good fit, you can purchase a Toolkit license to eliminate trial dialog pop-ups.

Hardware Key

When a license is activated for a product, the system information that identifies the installation hardware is contained within an encrypted string and is used to generate the license key for the product. The string containing the encrypted hardware information is known as the Hardware Key.

License Configuration File

When a Toolkit or an Automatically Reported Runtime (Node-Locked) license is purchased, a configuration file is provided that contains information about the license that was purchased. This file is used by the licensing utilities (License Manager and Server Licensing Utility (SLU)) to install a license on the system.

License Key

Each product license has a unique key associated with it that uniquely identifies the customer, product, version, edition, and platforms, and, in some cases, the hardware for which the license is valid.

License Manager

The License Manager is a GUI application that is used by developers to register and activate Toolkit licenses on their development systems.

Manually Reported Runtime Licensing

Runtimes may be licensed in one of two ways: automatic reporting or manual reporting. With Manually Reported Runtime (Non-Node-Locked), you embed all of the licensing information directly into your application. You must manually provide royalty reporting to Accusoft for the actual licenses deployed. This model will be the best choice for you if you are not connected to the Internet at runtime, as might be the case in an defense or financial application.

Node-Locked Licensing

Another name for Automatically Reported Runtime (Node-Locked).

Non-Node-Locked Licensing

Another name for Manually Reported Runtime (Non-Node-Locked).

Paid License

A paid license is a Toolkit license that you have purchased from Accusoft. It is ready to be used in production, with no trial dialog pop-ups. If you wish to test other editions of the Accusoft product for which you have purchased a Paid License, you may activate an Evaluation license for that edition; a complete set of Evaluation licenses for all editions is registered at installation time.

Registration

Registration is the process of informing the License Manager about a new Toolkit license you have purchased. It uses this information to create a license key on your system; this allows you to activate the product for development with no restrictions, such as trial dialog pop-ups.

Runtime (Deployment) Licensing

When deploying an application, a Runtime License is also required for each user’s installation. You must purchase runtime licenses, which are consumed as licenses are registered. There are two Runtime Licensing models: Automatically Reported Runtime (Node-Locked); and Manually Reported Runtime (Non-Node-Locked).

Server Licensing Utility (SLU)

The Server Licensing Utility (SLU) is a small GUI application that allows you to request a license key from the Runtime licenses you have purchased. The SLU is the mechanism for Automatically Reported Runtime (Node-Locked).

Toolkit Licensing

Each Toolkit is assigned to a specific developer who has a registered account with Accusoft. When the developer installs the Toolkit, it is not fully functional until they have activated the license through the use of the License Manager application. When the developer starts the License Manager, they are required to provide login credentials that identify and allow them to activate their license on a development system. See Toolkit Licensing.

Web Services

When a license is registered, information that uniquely identifies the customer’s or user’s hardware is passed along with the license information over the Internet to a web service. The web service validates the licensing request, generates a key that includes the hardware information, and returns the new license key to the application that made the licensing request. The license key is then stored on the requested system and used by the component when it is executed. It is also possible to register systems that are not directly connected to the Internet through the use of removable media and a different system that is connected to the Internet.

1.2.3 Getting Started

The ImageGear toolkit contains a comprehensive sample application program, which you can load at the same time you install ImageGear's program files. The sample is provided in source form. Accusoft permits you to use the source module of the sample program whole or to cut and paste from it as you wish, to create and expand your own applications. The sample included with this software has a variety of functions and is an especially good tool for seeing ImageGear's image processing functions. It allows the user to load an image and manipulate it in several ways. For example the user could rotate the image any number of degrees, apply image processing functions, or resize to any size.

This section provides information about the following:

- ImageGear Samples
- Developing an Application
  - Loading an Image
  - Displaying an Image
  - Changing Image Display Settings
    - Fit Mode
    - Align Mode
    - Image Orientation
    - Zooming an Image
  - Image Processing
    - Rotating an Image
    - Flipping an Image
- Using the Sample Code For Your Application

1.2.3.1 ImageGear Samples

Your ImageGear Professional toolkit contains ImageGearDemo Cocoa sample for Xcode IDE that is installed together with ImageGear shared object libraries and related files. The sample is provided in source form. You can compile and execute the application to see the actual effects of the ImageGear function calls within it, or examine the source file to find examples of ImageGear API function calls and accompanying platform-specific calls. In addition, Accusoft permits you to use the sample application's source code. You can use the sample code as is, or cut and paste from it as you wish, to create, then expand, your own applications. The only restriction is that you may not distribute the original ImageGear sample applications with your applications.

The sample lets you load, display, process, and save raster and PDF images. It also demonstrates clipboard operations. By running this application and then examining its source code, you can see how the calls to ImageGear functions such as **IG_load_...()**, **IG_dspl_...()**, **IG_IP_...()**, **IG_clipboard_...()** and **IG_fltr_...()** interact with the requirements and procedures of Mac OS X programming. You can later use this program as a template for developing an application of your own.

The sample is located in the directory to which you have installed ImageGear (see the sections Directory Structure and Description of Installed Files). You can begin developing an application easily by choosing the sample to start with as a template, making a backup copy to preserve it, and editing it - cutting and pasting whole sections from other samples, if you wish. In this way, you will begin with a working program that displays images on your screen from the start, so you can test and debug each new feature as you add it to your code.

1.2.3.2 Developing an Application

Accusoft provides the ImageGearDemo sample; use it as a template, and make a backup copy to preserve it. You may cut and paste whole sections from the original code. This way you begin with a working program that displays images on your screen from the start. You can easily add, test, and debug each new feature as you add it to your code. This section is set up as a tutorial that uses the sample application as a guide.

This tutorial walks you through several operations performed on a sample image:

- Loading an Image
- Displaying an Image
- Changing Image Display Settings
  - Fit Mode
  - Align Mode
  - Image Orientation
  - Zooming an Image
- Image Processing
  - Rotating an Image
  - Flipping an Image

In the process of following along with this tutorial you will be introduced to a few of ImageGear's many defined constants, which provide flexibility to ImageGear's functions.

To work along with the tutorial, access the sample on your system: open your sample folder and double-click the sample application.

1.2.3.2.1 Loading an Image

1. Choose **File** > **Open...**.

   The key ImageGear call used is IG_load_file(). We use NSOpenPanel dialog to select an image file. All related calls are encapsulated in the mnuFileOpen action in our Sample. Please see NSOpenPanel Class Reference for details.

2. The image will be displayed.

   Here is the segment of code that demonstrates this operation:

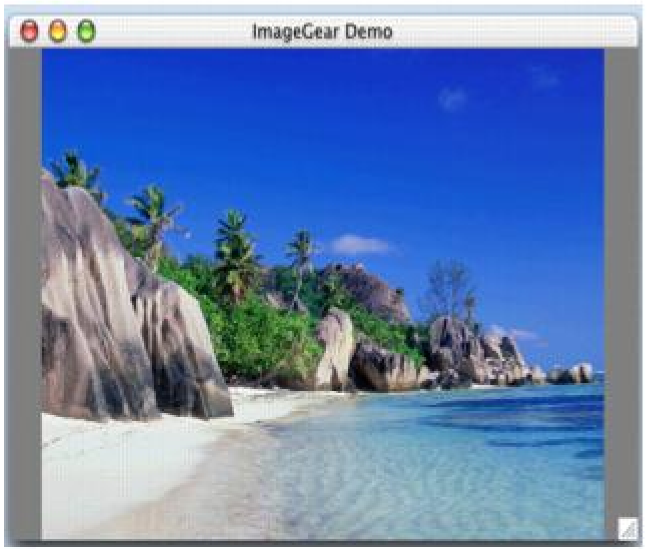
```
- (IBAction)mnuFileOpen:(id)sender {
    AT_ERRCOUNT errCount = 0;
    // Create the File Open Dialog class.
    NSOpenPanel* openDlg = [NSOpenPanel openPanel];

    // Enable the selection of files in the dialog.
    [openDlg setCanChooseFiles:YES];

    // Disable the selection of directories in the dialog.
    [openDlg setCanChooseDirectories:NO];

    // Display the dialog.  If the OK button was pressed,
    // process the files.
    if ( [openDlg runModal] == NSOKButton )
    {
        // Get file name as char*
        NSArray* URLs = [openDlg URLs];
        NSURL* URL = [URLs objectAtIndex:0];
        NSString* filePath = [URL path];
        const char* utf8FileName = [filePath UTF8String];
        // Delete an existing hIgear
        if(IG_image_is_valid(hIGear))
            IG_image_delete(hIGear);

        errCount = IG_load_file((LPSTR)utf8FileName, &hIGear);
        if(errCount != 0)
            printf("IG_load_file error:\n");

        // Update main view
        [mainScrollViewOutlet setNeedsDisplay:YES];
    }
}
```

1.2.3.2.2 Displaying an Image

As soon an image is loaded it can be displayed on screen. The simplest way to do this is to
use the IG_dspl_image_draw() function. In our sample this function is called in drawRect method of main view class.

```
- (void)drawRect:(NSRect)dirtyRect
{
    if(IG_image_is_valid(hIGear))
    {
        // Get device context
        CGContextRef myContext = [[NSGraphicsContext currentContext]
graphicsPort];
        if([NSGraphicsContext currentContextDrawingToScreen])
            // Draw the image to the screen
            IG_dspl_image_draw(hIGear, 0, (__bridge HWND)self, (HDC)myContext,
NULL);
        else
            // Print the image
            IG_dspl_image_print(hIGear, 0, (HDC)myContext, FALSE);
    }
}
```

Here is the sample application with a 24-bit image loaded:



1.2.3.2.3 Changing Image Display Settings

ImageGear provides API that specifies how an image is displayed on screen. It does not change the image itself; it only affects the image appearance in the display.

The following sections demonstrate some of the image display settings available:

- Fit Mode
- Align Mode
- Image Orientation
- Zooming an Image

1.2.3.2.3.1 Fit Mode

The sample contains a control for setting the Fit Mode. This method determines how the image is fitted in the window.

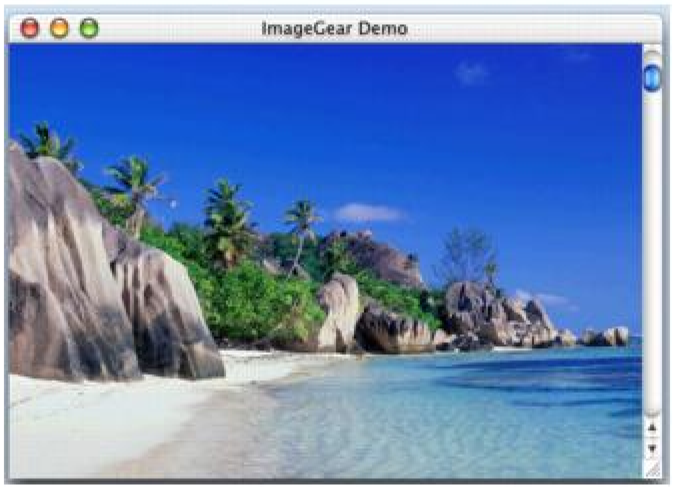
- Choose **Display** > **Layout** > **Fit** to display the image in one of the following Fit Modes:
  - **FIT_TO_WINDOW** - view the entire image in the window. The constant IG_DSPL_FIT_TO_DEVICE, which is defined in dspl.h file, is used as an argument in IG_dspl_layout_set() in order to fit the image to the window.
  - **FIT_TO_WIDTH** - view the image displayed as wide as the window. If the fit method is set to IG_DSPL_FIT_TO_WIDTH, the image is displayed as wide as the window, but may extend beyond the window vertically.
  - **FIT_TO_HEIGHT** - view the image displayed as high as the window. If the fit method is set to IG_DSPL_FIT_TO_HEIGHT, the image fits the height of the window, but may extend beyond the width boundaries.
  - **ACTUAL_SIZE** – the image in its actual size. If the fit method is set to IG_DSPL_ACTUAL_SIZE, the image is scaled 1:1.

Initially the fit method is set to IG_DSPL_FIT_TO_DEVICE, which means that the entire image are displayed without scrolling in the window. If you do not change this setting and proceed to change an image, doing a rotate for example, the image still retains the fit to window setting; you will still be able to see the entire image.

The following code example sets the fit mode to the **FIT_TO_WIDTH** value:

```
- (IBAction)mnuDisplayLayoutFitTO_WIDTH:(id)sender {
    if(IG_image_is_valid(hIGear))
    {
        // Reset zoom
        IG_dspl_zoom_set( hIGear, 0,
IG_DSPL_ZOOM_H_NOT_FIXED|IG_DSPL_ZOOM_V_NOT_FIXED, 1.0, 1.0 );
        // Set fit mode to IG_DSPL_FIT_TO_WIDTH
        IG_dspl_layout_set( hIGear, 0, IG_DSPL_FIT_MODE, NULL, NULL, NULL,
                           IG_DSPL_FIT_TO_WIDTH, 0, 0, 0.0 );
        // Update main view
        [mainScrollViewOutlet setNeedsDisplay:YES];
    }
}
```

In the sample below the image is fit to the width of the screen:



The scroll bar automatically appears on the right side of the image because the image is now as wide as the screen but is longer than the screen.

1.2.3.2.3.2 Align Mode

Another display attribute is Align Mode. It determines where the image is displayed on the screen: in the center or aligned to the one of borders.

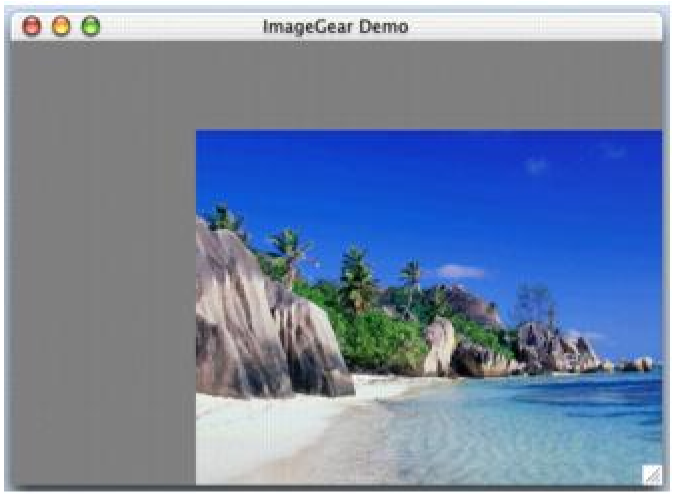
- Choose **Display** > **Layout** > **Align** to display the image in one of the following Align Modes:
  - Vertically, you can choose to align to the top, center, or bottom.
  - Horizontally, you can choose to align to the left, center, or right.

The mode can be specified with the IG_DSPL_ALIGN_ constants declared in dspl.h file.

The following code example specifies that the image will be displayed in the right-bottom corner of the screen:

```
- (IBAction)mnuDisplayLayoutAlignRIGHT_BOTTOM:(id)sender {
    if(IG_image_is_valid(hIGear))
    {
        // Set align mode to right-bottom
        IG_dspl_layout_set( hIGear, 0, IG_DSPL_ALIGN_MODE, NULL, NULL, NULL,
                            0, IG_DSPL_ALIGN_X_RIGHT|IG_DSPL_ALIGN_Y_BOTTOM, 0,
0.0 );
        // Update main view
        [mainScrollViewOutlet setNeedsDisplay:YES];
    }
}
```

In the sample below the image is aligned to right and bottom:



See Also

IG_dspl_layout_set()

1.2.3.2.3.3 Image Orientation

Orientation parameters allow you to rotate and flip an image on screen without changing the image bitmap.

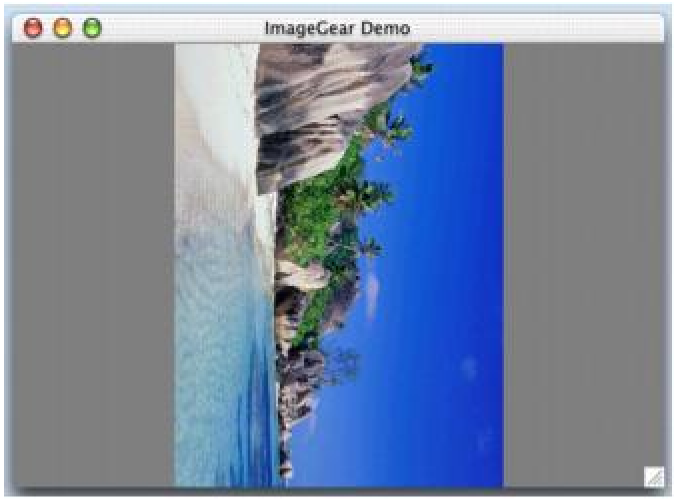
- Choose **Display** > **Layout** > **Orientation**. Displayed image orientation can be specified by 2 parameters:
  - Where the image's top side is located (relative screen orientation).
  - Where the image's left side is located.

Normal image orientation is top on top and left on left (i.e., image top on screen top, and image left on screen left), which can be specified with the IG_DSPL_ORIENT_TOP_LEFT constant.

IG_DSPL_ORIENT_RIGHT_TOP (i.e., top on right and left on top) constant effectively rotates the image by 90 degrees clockwise. This can be done with the following code fragment:

```
- (IBAction)mnuDisplayOrientRIGHT_TOP:(id)sender {
    if(IG_image_is_valid(hIGear))
    {
        // Set display orientation to IG_DSPL_ORIENT_RIGHT_TOP
        IG_dspl_orientation_set( hIGear, 0, IG_DSPL_ORIENT_RIGHT_TOP);
        // Update main view
        [mainScrollViewOutlet setNeedsDisplay:YES];
    }
}
```

The sample below demonstrates the image oriented to right-and-top (IG_DSPL_ORIENT_RIGHT_TOP):



See Also

IG_dspl_orientation_set

1.2.3.2.3.4 Zooming an Image

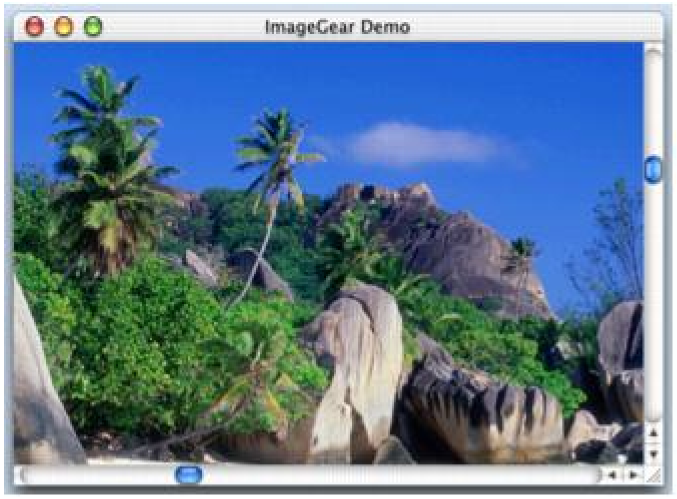
From the **Display** menu choose **Zoom in** to zoom an image in or **Zoom out** to zoom an image out.

The following code fragment demonstrates a Zoom in operation:

```
- (IBAction)mnuDisplayZoomIn:(id)sender {
    double dblHZoom, dblVZoom;

    // Get previous zoom factors
    IG_dspl_zoom_get(hIGear, 0, (__bridge HWND)mainScrollViewOutlet, NULL,
&dblHZoom, &dblVZoom );
    if( dblHZoom <= 10 && dblVZoom <= 10)
    {
        dblHZoom *= 1.25;
        dblVZoom *= 1.25;
        // Set new zoom factors
        IG_dspl_zoom_set( hIGear, 0, IG_DSPL_ZOOM_H_FIXED|IG_DSPL_ZOOM_V_FIXED,
dblHZoom, dblVZoom );
    }
    // Update main view
    [mainScrollViewOutlet setNeedsDisplay:YES];
}
```

IG_dspl_zoom_get() is used to obtain current image zoom settings. In the next line, the current settings were increased by 25%, and the new zoom value was set using IG_dspl_zoom_set().

The example below is the original image zoomed in several times:



When using the Zoom command, scroll bars automatically appear on the bottom and on the right side because the image no longer fits in the window.

Scrollbars do not automatically appear until you display them with IG_dspl_scroll_set() call. See also InitScrollBars() function in the ImageGearDemo Sample.

1.2.3.2.4 Image Processing

ImageGear provides many options for Image Processing transformations: rotating, flipping, cropping and resizing of images, color reducing and promoting, etc. See Processing Images for more detailed information about Image Processing functionality. This tutorial provides information about the following functionality:

- Rotating an Image
- Flipping an Image

1.2.3.2.4.1 Rotating an Image

The **Processing** menu contains a group of **Rotate 90** and **Rotate Any** items.

Using IG_IP_rotate_multiple_90(), you can rotate your image on 90, 180, or 270 degrees. The code sample below demonstrates a 90 degree rotation. The required angle of rotation should be specified by one of IG_ROTATE_ constants.

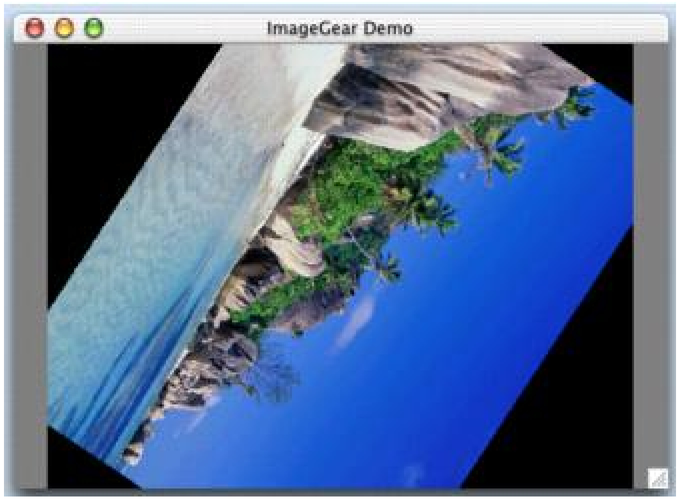
```
- (IBAction)mnuProcessingRotate90:(id)sender {
    if(IG_image_is_valid(hIGear))
    {
        IG_IP_rotate_multiple_90(hIGear, IG_ROTATE_90);
        // Update main view
        [mainScrollViewOutlet setNeedsDisplay:YES];
    }
}
```

Although the visible result of this function call is similar to that of IG_dspl_orientation_set() described in Image Orientation, the significant difference between them is that all functions of the ImageGear Image Processing group (IG_IP_) do change image's contents, while the ImageGear Image Display functions (IG_dspl_) do not.

IG_IP_rotate_any_angle() allows you to rotate an image to any angle. In the sample below, the image will be rotated 123.5 degrees.

```
- (IBAction)mnuProcessingRotateAny:(id)sender {
    if(IG_image_is_valid(hIGear))
    {
        IG_IP_rotate_any_angle(hIGear, 123.5, IG_ROTATE_CLIP);
        // Update main view
        [mainScrollViewOutlet setNeedsDisplay:YES];
    }
}
```

Below you can see the image that is rotated on 123.5 degrees:



1.2.3.2.4.2 Flipping an Image

The **Processing** menu contains a group of **Flip Vertically** and **Flip Horizontally** items.

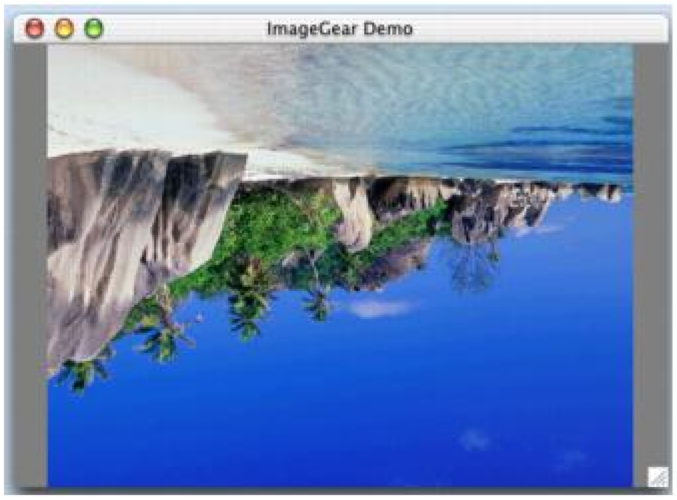
The following code fragment demonstrates a vertical flip:

```
- (IBAction)mnuProcessingFlipV:(id)sender {
     if(IG_image_is_valid(hIGear))
     {
          IG_IP_flip(hIGear, IG_FLIP_VERTICAL);
          // Update main view
          [mainScrollViewOutlet setNeedsDisplay:YES];
     }
}
```

Similarly, using the IG_FLIP_HORIZONTAL constant for the last argument of IG_IP_flip() flips an image horizontally.

As in the case of Rotating an Image, this function physically rearranges the pixels of your source image.

This is an original image flipped vertically:



1.2.3.3 Using the Sample Code For Your Application

If you want your applications to use any of the features available in the sample, you can cut and paste the source code from the sample right into your application. ImageGearDemo sample contains all of the code for the sample application. For more information on the functions referenced in this tutorial, see Using ImageGear and the Core Component API Function Reference.

1.2.4 Using ImageGear

This chapter describes how to use ImageGear's functions to fulfill your applications' imaging needs. Examples and recommendations for using ImageGear functions can be found in the API Reference chapters as well as in the Chapter.

1.2.4.1 General Aspects

This section provides information about the following:

1.2.4.1.1 ImageGear Architecture Overview

ImageGear API is a set of C functions, callback function declarations, structures, enumerations, and macros.

The central element in ImageGear API is the single page image handle: HIGEAR. It can contain raster or vector data for a single image page. The majority of ImageGear API functions take HIGEAR as a parameter. See Single-Page Images for more information.

ImageGear also allows working with multi-page documents. See Multi-Page Documents for more information.

The most common way to obtain a HIGEAR is to load a page from an image file, located on a disk, in a memory buffer, or at an Internet location. You can also create a blank HIGEAR or paste an image from clipboard. See the following sections for details:

- Loading Images
- Clipboard Operations

Once the image is in memory, you can process it in a variety of ways, for example: save to a disk file, a memory buffer, or an internet location, display, print, apply image processing operations, convert the image to a different pixel format, access image pixels directly, annotate the image, or recognize text in the image. See the following sections for details:

- Saving Images
- Displaying Images
- Processing Images
- Color Management
- Accessing Image Pixels
- Annotating Images

ImageGear also provides advanced metadata (non-image data) support, including support for TIFF, EXIF, XMP, IPTC, Photoshop, and other types of metadata. Note, however, that ImageGear does not store metadata with a HIGEAR or HMIGEAR handle, but instead provides callbacks to get or set (add) metadata during image loading and saving. It is the application's responsibility to store the metadata between loading and saving an image. See Non-Image Data Processing for details.

Several image file formats have features that are not used by other file formats. ImageGear provides specialized API for these formats. Please see Advanced Image Formats for details.

See these additional topics for information on other important aspects of ImageGear API:

- Error Detection and Handling
- ImageGear Components
- Thread Safety

1.2.4.1.2 API Naming Conventions

ImageGear Core Component functions are named in accordance with their purpose within the Core Component API Function Reference.

With the exception of callback functions, which always begin with the prefix "LPFNIG", the function names adhere to the following conventions:

- All function names begin with the prefix "IG_" (always in uppercase). This is to identify them as ImageGear functions, and to avoid conflicts with your application's functions.
- Following "IG_" is the name of the group to which the function belongs.

The table below displays ImageGear function groups' name and purpose:

Function Name	**Function Purpose**
IG_clipboard_ ...()	Clipboard
IG_ ... _CB_ ...() or LPFNIG_ ...()_CB_	Callback
IG_colorspace_conversion_ ...()	Color space conversion
IG_comm_ ...()	Component manager
IG_cpm_ ...()	Color profile manager
IG_DIB_ ...()	DIB Info Services
IG_display_ ...()	Display
IG_dspl_ ...()	Displaying and printing
IG_error_ ...(), IG_version_ ...(), IG_err_...()	Library utility
IG_fltr_ ...()	Format Filter processing
IG_FX_ ...()	Special Effects
IG_image_ ...() or IG_palette_ ...()	Image utility
IG_IP_ ...()	Image processing
IG_gctrl_ ...()	Global control parameters processing
IG_GUI_ ...()	Graphical user interface
IG_load_ ...()	Loading
IG_mult_ ...()	Multimedia
IG_pixel_ ...()	Pixel access
IG_save_ ...()	Saving
IG_thread_ ...()	Thread safety
IG_util_colorspace_ ...()	Color space utility
IG_vector_...()	Vector utility

1.2.4.1.3 Error Detection and Handling

ImageGear functions handle all errors in a single, uniform way. Even low-level ImageGear functions that you cannot call directly handle errors in this same way.

When an error condition is detected by an ImageGear function, the function places an error code indicating specifically what happened, along with information about where the error occurred, in an internal memory area called the ImageGear error stack. This error stack remains available to your application as it executes, so you can inspect and treat the errors where (in your program code) needed. After placing the error or errors on the error stack, the ImageGear function returns to its caller (returning the count of errors now on the stack, if the function's return type is AT_ERRCOUNT).

More than one error may be placed on the error stack as a result of a single ImageGear function call. This is because an ImageGear function that you call will often call lower-level ImageGear functions (not directly callable by your application). Each such lower level function may itself place an error onto the error stack before returning to its caller. Upon return to your application, there may be several errors on the stack. Note that in such a case, the lowest level function's error was placed on the stack first, and the highest level function (the one that your application called directly) placed its error on the stack last, just before returning to your application, because it could not proceed (due to the error).

ImageGear provides the following general functions for accessing the error stack:

IG_err_stack_clear	Deletes all records (errors and warnings) from the error stack.
IG_err_count_get	Returns the total number of records (errors plus warnings) on the error stack.
IG_err_error_check	Returns the number of records of the specified level (either errors or warnings) on the error stack.
IG_err_error_get	Retrieves information about a record on the specified level of the error stack. Use this function if you are only interested in errors but not in warnings, or only warnings and not errors.
IG_err_record_get	Obtains information about a record on the error stack. Use this function if you are interested in both errors and warnings.
IG_err_error_set	Places a record onto the error stack.

The following additional functions are available:

IG_error_check	Returns the number of errors currently on the error stack.
IG_error_clear	Despite its name, clears both errors and warnings from the stack. Same as IG_err_stack_clear.
IG_error_get	Retrieves information about an error from the error stack.
IG_error_set	Places an error record onto the error stack.
IG_warning_check	Returns the number of warnings currently on the ImageGear error stack.
IG_warning_clear	Clears all warnings from the error stack.
IG_warning_get	Retrieves an ImageGear warning Code and associated information from the error stack.
IG_warning_set	Places an ImageGear warning onto the error stack.

The following functions provide access to error callback functions and data:

IG_err_callback_get	Obtains error stack callback data and functions that are called to signal error stack changes for the current thread.
IG_err_callback_set	Sets error stack callback data and functions that are called to signal error stack changes for the current thread.

IG_errmngr_callback_get	Obtains error stack callback data and functions that are called to signal error stack changes for all threads.
IG_errmngr_callback_set	Sets error stack callback data and functions that are called to signal error stack changes for all threads.

The example below shows how you may get information about all errors on the error stack using IG_err_error_get function. Refer to a description of this function in the Core Component API Function Reference for thorough explanations of its arguments.

```
HIGEAR hIGear = 0;               // Will hold the handle returned by IG_load_file
AT_ERRCOUNT nErrCount;           // Count of errors on the stack upon function return

// Load image file "picture_bad.bmp" from working directory
// and obtain the image's HIGEAR handle:
nErrCount = IG_load_file ( "picture_bad.bmp" , &hIGear );
if(nErrCount != 0)
{
        // Get all errors and report them
        AT_INT i;
        CHAR szFileName[MAX_PATH]; // ImageGear source file name where the error occurred
        INT nLineNumber; // Line number where the error occurred
        AT_ERRCODE nErrCode; // Error code
        AT_INT nValue1; // First value associated with the error
        AT_INT nValue2; // Second value associated with the error
        CHAR szExtraText[1024]; // Text description of the error
        for(i = 0; i < nErrCount; i ++)
        {
                IG_err_error_get(0, (UINT)i, szFileName, (UINT)sizeof(szFileName),
                        &lineNumber, &nErrCode, &nValue1, &nValue2, szExtraText,
(UINT)sizeof(szExtraText));
                        // Process the error information
                        //...
        }
}
else
{
        //...
        // Destroy the image
        if(IG_image_is_valid(hIGear))
        {
                IG_image_delete(hIGear);
        }
}
```

Each ImageGear function, excluding all IG_dspl_...() functions, clears the error stack upon entry. Therefore, after an ImageGear function call you should check the stack prior to your next ImageGear function call.

1.2.4.1.4 ImageGear Components

ImageGear uses a component structure that consists of the Core (main) ImageGear component and a number of additional components.

A component is a module that can be connected to the main ImageGear module using a platform-independent API. To initialize the component functionality, attach the component to the main ImageGear module using the ImageGear component manager API. When the component is attached, it gets access to all core ImageGear functions, and the core ImageGear functionality can use the component's structures, functions, and control parameters.

Usually, ImageGear components contain functionality such as additional format filters (for example, LZW or PDF) or additional image processing functionality (such as ImageClean, ART).

Once you have attached the component, it cannot be detached prior to unloading of the ImageGear Core module. All components are detached and unloaded automatically at the time of the ImageGear Core module unloading.

ImageGear identifies every component by its name, and during the attachment process, it calculates the physical name of the file where the component is located. By default ImageGear assumes that all components are located in the same directory where the main ImageGear module is located, however you can specify a different folder from which to load the components.

This section provides information about the following:

- ImageGear Component Descriptions
    - ImageGear Core Component
    - ImageGear GIF/TIFF-LZW Component
    - ImageGear Medical Component
    - ImageGear PDF Component
- Component Manager API
- Calling ImageGear Component API Functions
- ImageGear Component Names

1.2.4.1.4.1 ImageGear Component Descriptions

The following sections summarize the ImageGear Professional components that are currently available. For more information on these components, see the documentation pertaining to each one.

- ImageGear Core Component
- ImageGear GIF/TIFF-LZW Component
- ImageGear Medical Component
- ImageGear PDF Component

1.2.4.1.4.1.1 ImageGear Core Component

The ImageGear Core Component provides the ImageGear Core Component API.

1.2.4.1.4.1.2 ImageGear GIF/TIFF-LZW Component

This component allows you to work with GIF/TIFF-LZW compression images.

See Also:

GIF Non-image Data Structure

GIF File Format

LZW (Lempel-Ziv-Welch) Compression

1.2.4.1.4.1.3 ImageGear Medical Component

The ImageGear Medical (MD) component is a full-featured ImageGear component that supports the DICOM format, contains a custom API, and includes expanded image processing capabilities beyond those of the baseline ImageGear library.

The format support of the MD component includes loading and saving monochrome, palletized, and true color medical images using the following file formats:

- DICOM 3.0 Part 10-compliant images
- DICOM 3.0 Raw Format (non-Part 10-compliant)

In addition, your application will continue to support all ImageGear-supported file formats, allowing you to convert an image of a different format to a medical image format, and vice-versa.

See Also:

Advanced Image Formats > DICOM

MD Component API Reference

1.2.4.1.4.1.4 ImageGear PDF Component

The ImageGear PDF Component allows you to load, save, and process Adobe PDF (Portable Document Format) file format images using Core ImageGear and other ImageGear Component functionality.

In addition, this PDF Component provides the ability to extract text from loaded PDF image files.

PostScript format is not supported on MacOS X platform.

See Also:

Advanced Image Formats > Adobe PDF

PDF Component API Reference

1.2.4.1.4.2 Component Manager API

ImageGear provides the component manager API for attaching, checking, function requesting, and retrieving information of every ImageGear component.

To attach (load) the component to the core ImageGear you can use the function:

```
IG_comm_comp_attach(LPCHAR lpCompName)
```

This function attaches the component, determined by lpCompName (e.g., "PDF"), to the core ImageGear module. If a component with the specified name is already attached, this function does nothing. See also the ImageGear Component Names section.

To check if a component is already attached, use the function:

```
IG_comm_comp_check(LPCHAR lpCompName)
```

If the component is attached, this function returns TRUE; it returns FALSE otherwise.

The following two functions are used for calling component API functions. See more details on using these functions in Calling ImageGear Component API Functions.

```
IG_comm_function_call(LPCHAR lpEntryName, ... );
```

```
IG_comm_entry_request(
       LPCHAR lpEntryName,
       LPAFT_ANY *lpFuncPtr,
       LPCHAR lpReason
);
```

The following function retrieves information about all attached components:

```
IG_comm_comp_list(
       LPUINT *lpnCount,
       UINT nIndex,
       LPCHAR lpComp,
       DWORD dwCompSize,
       LPUINT lpnRevMajor,
       LPUINT lpnRevMinor,
       LPUINT lpnRevUpdate,
       LPCHAR lpBuildDate,
       UINT nBDSize,
       LPCHAR lpInfoStr,
       UINT nISSize
);
```

It returns the number of attached components, and the complete information about the component specified by nIndex.

1.2.4.1.4.3 Calling ImageGear Component API Functions

As mentioned in the previous section, some components only provide support for additional file formats, and don't expose any API functions, and some components provide additional API. If your application uses a component that provides additional API functions, there are a few additional steps needed to use these API functions.

There are two ways that the application can call a function implemented by the component. The first method is calling a component's API functions via a component manager function:

```
IG_comm_function_call(LPCHAR lpEntryName, ... );
```

where lpEntryName is a name of the requested function in the form "<COMP_NAME>.<FUNC_NAME>", where <COMP_NAME> is a name of the component that provides the function, and <FUNC_NAME> is the name of the function.

To simplify the calling of component functions, all components provide special macros for each of their public functions. This macro is located in the header file i_<COMP_NAME>.h for each component. For instance, i_CLN.h for ImageClean component:

```
#define IG_IC_clean_borders_ex( hIGear, nLeftBorderSize, nRightBorderSize,
nTopBorderSize, nBottomBorderSize, nMinLinesNum, nMinLineWidth)
    ((AT_ERRCOUNT (CACCUAPI *)(LPCHAR, HIGEAR, UINT, UINT, UINT, UINT, UINT, UINT))
IG_comm_function_call)("CLN.IG_IC_clean_borders_ex",  hIGear, nLeftBorderSize,
nRightBorderSize,  nTopBorderSize,  nBottomBorderSize,  nMinLinesNum,
nMinLineWidth)
```

With the use of this macro, a call to a component function looks exactly like a call to a regular C function:

```
#include "i_CLN.h"
…
IG_IC_clean_borders_ex( hIGear, nLeftBorderSize, nRightBorderSize,
nTopBorderSize, nBottomBorderSize, nMinLinesNum, nMinLineWidth);
```

Another method is to obtain a pointer to the component function and then call this function via its pointer. Use the following function to obtain a component function pointer:

```
IG_comm_entry_request(
        LPCHAR lpEntryName,
        LPAFT_ANY *lpFuncPtr,
        LPCHAR lpReason
);
```

- The first parameter is the name of the function in the format described above.
- The second parameter is a pointer a variable of type LPAFT_ANY, which will be overwritten with the pointer to the necessary function.
- The third parameter is a text description reason to get access for this function (it is optional and can be NULL).

The component public header contains a type declaration for all its public functions. The correct way to call such a function would be to declare the variable of the necessary function type defined in the component's public header; use IG_comm_entry_request() to initialize this variable with the correct value and then call it.

Calling component API functions by their pointers provides better performance, because it avoids the overhead of finding a function pointer by its name. If your application does not call component functions repeatedly in time-critical routines, you can use the simple method of calling component functions via their macros.

1.2.4.1.4.4 ImageGear Component Names

Below is the list of ImageGear components and their short names that you should provide for IG_comm_comp_attach() through the lpCompName argument:

- ImageGear Core Component - "CORE"
- ImageGear LZW Component - "LZW"
- ImageGear Medical Component - "MED"
- ImageGear PDF Component - "PDF"

1.2.4.1.5 Thread Safety

ImageGear and its associated components are completely thread-safe. The implementation of thread safety in ImageGear maximizes the performance of threaded applications on multi-CPU computers. Every ImageGear API function can be executed within a thread.

Five thread safety APIs are explained in detail in the Core Component API Function Reference.

- IG_thread_data_ID_associate() provides thread customized ImageGear settings.
- IG_thread_data_ID_get() provides thread customized ImageGear settings.
- IG_thread_local_data_cleanup() provides thread customized ImageGear settings.
- IG_thread_image_lock() is required in the situation where several threads are accessing the same HIGEAR concurrently AND at least one of these threads performs an operation that modifies/deletes HIGEAR.
- IG_thread_image_unlock() is required in the situation where several threads are accessing the same HIGEAR concurrently AND at least one of these threads performs an operation that modifies/deletes HIGEAR.

In most cases, no additional API calls are required to achieve thread safety.

Access to the same PDF document from multiple threads is not permitted, because multiple threads cannot share Adobe PDF Library data types. PDF docs created/opened in the main thread can be only used from the main thread.

1.2.4.1.6 Global Control Parameters

ImageGear provides a set of API functions that allow you to add new global control parameters to your ImageGear application, set new values for existing global control parameters, and retrieve information about these parameters.

To add a new global parameter as well as to set new values for existing parameters, use the function IG_gctrl_item_set:

```
IG_gctrl_item_set(
        LPCHAR ControlID,
        AT_MODE nValueType,
        LPVOID lpValue,
        DWORD dwValueSize,
        LPCHAR lpTextInfo
);
```

This function will search for the global parameter specified by the ControlIDname (syntax: "<GRPNAME>.<Param name>"), and if it is found will set a new value for it. If it is not found, the function will add this new parameter to the global control parameters list. Through the lpTextInfo argument, you can also set the text description of the specified global parameter.

To retrieve information about the global control parameter, use the functions IG_gctrl_item_get and IG_gctrl_item_by_index_get:

```
IG_gctrl_item_get(
        LPCHAR CtrlID,
        LPAT_MODE lpnValType,
        LPVOID lpValue,
        DWORD dwValSize,
        LPDWORD lpdwValSize,
        LPCHAR lpTextInfo,
        DWORD dwTextBufSize,
        LPDWORD lpdwTextInfoSize
);

IG_gctrl_item_by_index_get(
        UINT nIndex,
        LPCHAR CtrlID,
        DWORD dwIDSize,
        LPAT_MODE lpnValType,
        LPVOID lpValue,
        DWORD dwValSize,
        LPDWORD lpdwValSize,
        LPCHAR lpTextInfo,
        DWORD dwTextBufSize,
        LPDWORD lpdwTextInfoSize
);
```

The first function returns the value and the text description of the global control parameter specified by name. The second function returns information about the control parameter specified by its index in the global parameters list. Both functions return FALSE if the specified global parameter is not found.

If you want to know the general amount of global control parameters currently existing in the global parameters list, call the function IG_gctrl_item_count_get:

```
IG_gctrl_item_count_get();
```

If you need to know an index of the global control parameter in the parameters array, use this function IG_gctrl_item_id_get:

```
IG_gctrl_item_id_get (
        UINT nIndex,
        LPCHAR lpCtrlID,
        UINT nBufSize
);
```

Please also see the list of all ImageGear Global Control Parameters.

1.2.4.1.7 Callback Functions

ImageGear provides callback function support for load, save, print, and other operations to enable your application to control these processes. Callback functions are functions for which you write the code, and whose names you provide to ImageGear. ImageGear will call them at appropriate breakpoints in an operation (such as after each raster line has been processed), at which time your function may modify image data, display status information, or perform other auxiliary operations specified before returning control to ImageGear.

This section discusses how to declare, code, and invoke an ImageGear callback function; how to register an ImageGear callback function; and how to work with the status bar and tag callback functions.

Callbacks are actually function types (or templates) where you can include your own code to carry out extra operations during normal ImageGear file processing. Callbacks can be passed data, return data, or both. All callback type names begin with the prefix "LPFNIG" which stands for "Long Pointer to a FuNction of ImageGear". Due to this unusual prefix, their descriptions can easily be found in the section Core Component Callback Functions Reference in the Core Component API Function Reference.

Some callback functions return a Boolean value to ImageGear, indicating whether you want ImageGear to continue the operation, disregard the instructions in the callback, or abort an operation. Most callbacks are VOID, exchanging their information through their arguments.

How a callback function is coded, declared to ImageGear, and invoked by ImageGear, is illustrated by the simple examples below in which an application calls function IG_load_file_display() to load and then display an image. IG_load_file_display() will automatically call your callback of type LPFNIG_LOAD_DISP.

The following example shows a call to register a display callback and the callback itself:

```
VOID ACCUAPI  my_set_attributes_func
  (
      LPVOID  lpPrivateData, /* Ptr to private data area      */
      HIGEAR  hIGear         /* Handle of loaded image         */
   )
{
/* This callback function disables centering of the image: */
IG_dspl_layout_set( hIGear, IG_GRP_DEFAULT, IG_DSPL_ALIGN_MODE, NULL, NULL, >NULL,
0,
IG_DSPL_ALIGN_X_LEFT|IG_DSPL_ALIGN_Y_TOP, 0, 0.0 );
return;
}
```

In response to the above call, ImageGear loads the image, creates a DIB and a HIGEAR data structure, and then calls your callback function. When your callback function returns, ImageGear will display the image, and then return to the statement following IG_load_file_display().

Note that your module containing the IG_load_file_display() should contain in its initial definitions a function prototype or declaration for the callback function. There are two ways that the callback can be declared:

- By prototype:

```
VOID ACCUAPI  my_set_attributes_func ( LPVOID lpPrivate, HIGEAR hIGear );
```

- By declaration:

```
LPFNIG_LOAD_DISP my_set_attributes_func;
```

If you do not write code for a callback function type that is part of a normal API call, such as IG_load_file_display(), you can just pass in a NULL for the callback parameter.

This section also provides information about the following:

- Private Data Use in Callback Functions
- Registering a Callback Function
- Status Bar Callback
- Using Filter Callback Functions to Process Non-Image Data

1.2.4.1.7.1 Private Data Use in Callback Functions

If you look at the argument lists of the ImageGear callback function types you will notice that many of them have an argument for holding private data. This can be used for anything you like. In some cases, the function which calls or registers the callback will also contain a parameter for private data which will be directly passed to your callback. IG_load_file_display() fits this description. In the following example, you will see how the fourth argument of this function can be used to pass private data to the callback for additional flexibility:

```
LPFNIG_LOAD_DISP   my_set_attributes_func;
HIGEAR          hIGear;
HDC             hDC;
DWORD           dwGroupID
HWND            hWnd
AT_ERRCOUNT     nErrcount;
AT_MODE         nAlignMode;     /* align mode  */
/* Instead of NULL, give the address where private data (&bCenter) begins:*/
nErrcount = IG_load_file_display ( "picture.bmp", hDC, dwGroupID, hWnd,
                my_set_attributes_func, &nAlignMode, &hIGear );
/* And the corresponding callback function:  */
VOID ACCUAPI  my_set_attributes_func (LPVOID lpPrivateData, HIGEAR hIGear)
{
/* Instead of FALSE, give the BOOL value located at the start of the private data
area: */
IG_dspl_layout_set( hIGear, IG_GRP_DEFAULT, IG_DSPL_ALIGN_MODE, NULL, NULL, NULL, 0,
(AT_MODE) *lpPrivateData, 0, 0, 0 );
>return;
}
```

You can pass any amount of private data that you would like. You can define a structure to hold your private data, and can provide the address of the structure (instead of the address of a single variable as in the above example).

1.2.4.1.7.2 Registering a Callback Function

You inform ImageGear of each callback function that you want it to call by specifying a pointer to that function as an argument in a call to an IG_...() function. The ImageGear functions that include at least one callback are:

- IG_load_file_display()
- IG_file_IO_register()
- IG_load_tag_CB_register()
- IG_save_tag_CB_register()
- IG_status_bar_CB_register()

Some callbacks are registered with special registration calls while others are passed in as arguments to normal API calls. The reason for this is that some callbacks need to be called during more than one ImageGear process. Callbacks of this nature will generally be registered with a special registering function (which includes the word "register") and called by ImageGear behind the scenes.

Other callbacks whose function is limited to just one API call will be passed in as an argument .

These callbacks are often essential to the completion of their host function .

If you've registered a callback function using an IG_ ..._CB_register() function, you can un-register it by calling the function again, and supplying a NULL in place of the pointer to the callback function.

In each case, the IG_...() function's description in Core Component API Function Reference chapter describes what type the callback function must be. This refers to the argument sequence with which ImageGear is going to call that particular callback function.

1.2.4.1.7.3 Status Bar Callback

ImageGear function IG_status_bar_CB_register() registers a callback function that will thereafter automatically be called by many ImageGear functions at the end of processing each raster line. The arguments let your callback function compute the completed percentage, so you can maintain and update a status bar or a message box showing this information.

Most of the following ImageGear functions will automatically call (not "register") a status bar callback function if you've registered it:

- IG_load_ ...()
- IG_save_ ...()
- IG_dspl_ ...()
- IG_IP_ ...()
- IG_FX_ ...()

To register your own status bar callback function (and the area you use if you want to pass your own data to it) make the following call:

```
IG_status_bar_CB_register ( my_status_bar_CB_func, &myPrivateData );
```

Your status bar callback function must be of type LPFNIG_STATUS_BAR. Whenever called by ImageGear, your status bar callback function will be called with the following argument list:

```
BOOL ACCUAPI my_status_bar_CB_func ( LPVOID lpPrivate, PIXPOS cYPos,
DIMENSION dwHeight );
```

Note the following:

- The function returns an AT_BOOL value, TRUE or FALSE. Return TRUE to have ImageGear proceed normally, return FALSE to tell ImageGear to put an IGE_INTERRUPTED_BY_USER error in the error stack and return from the IG_...() call it is processing (that is, to abort the operation returning the above-named error).
- The second argument indicates the raster line number just processed.
- The third argument indicates the total number of raster lines in the image. However, since some functions do not process the lines in order, the quantity (cYPos/dwHeight) in general will not tell you the fraction completed. Instead, your callback function should count the number of times it has been called since the operation began, and divide this count by dwHeight to obtain the fraction completed.

1.2.4.2 Images and Documents

This section provides information about the following:

1.2.4.2.1 Single-Page Images

The central element in ImageGear API is the single-page image handle: HIGEAR. The majority of ImageGear API functions take HIGEAR as a parameter. HIGEAR encapsulates the following data:

- DIB information, such as dimensions, color space, and channel depths. See DIB Information for more details.
- Image pixels (if HIGEAR contains a raster image). Usually, you do not need to access the image pixels directly. You can load, display, process, save images, and do other operations using high-level API that accesses image pixels internally. If you need to access image pixels directly, see Accessing Image Pixels for details.
- Image display attributes. See Displaying Images for details.
- Image orientation. See Image Orientation for details.
- Color profile (optional). See Using Color Profile Manager for details.
- Non-rectangular Area of Interest (optional). See Region of Interest Processing for details.
- Format-specific information (for DICOM).

Usually you create a HIGEAR handle by loading an image from a disk file, or from memory. See Loading Images for details. You can also create a new HIGEAR handle using IG_image_create, or import a Windows DIB into HIGEAR using IG_image_DIB_import.

When HIGEAR is no longer used, you must delete it using IG_image_delete.

See Also

Multi-Page Documents

1.2.4.2.1.1 DIB Information

ImageGear provides two ways to access image attributes: via the HIGEAR handle, and via the special object HIGDIBINFO, which encapsulates the image attributes.

Image Attribute	**To Access via HIGEAR**	**To Access via HIGDIBINFO**
Width	IG_image_dimensions_get	IG_DIB_width_get
Height	IG_image_dimensions_get	IG_DIB_height_get
Color Space	IG_image_colorspace_get	IG_DIB_colorspace_get
Channel Depths	IG_image_channel_count_get IG_image_channel_depth_get IG_image_channel_depths_get	IG_DIB_channel_count_get IG_DIB_channel_depth_get IG_DIB_channel_depths_get
Palette (for images that have Indexed colorspace)	IG_palette_get IG_palette_set	IG_DIB_palette_alloc IG_DIB_palette_length_get IG_DIB_palette_size_get IG_DIB_palette_pointer_get
Resolution	IG_image_resolution_get IG_image_resolution_set	IG_DIB_resolution_get IG_DIB_resolution_set
Signed attribute	IG_image_is_signed_get IG_image_is_signed_set	

The HIGDIBINFO object only contains the attributes and does not contain the pixels.

Use IG_DIB_info_create to create a DIB info object. Use IG_DIB_info_copy to create a copy of an existing object. Use IG_image_DIB_info_get to obtain DIB information from a HIGEAR handle.

When the HIGDIBINFO object is no longer in use, you must delete it using IG_DIB_info_delete.

You cannot edit the DIB information of a HIGEAR directly. Instead, use image processing functions to modify the image: resize, convert to a different color space, change its resolution, etc.

1.2.4.2.1.2 Image Orientation

ImageGear uses two different places to store an image's orientation information: Orientation attribute of HIGEAR, and Orientation attribute of display settings.

When ImageGear loads an image, and the image's file format supports storing the Orientation attribute (e.g., TIFF format), ImageGear stores this attribute in HIGEAR. You can get or modify this attribute with IG_image_orientation_get and IG_image_orientation_set, correspondingly.

If the application displays the image, ImageGear copies HIGEAR's Orientation attribute to the image's Display settings. This allows ImageGear to display the image using the orientation specified in the source file. You can get or set display orientation using IG_dspl_orientation_get and IG_dspl_orientation_set, correspondingly. Changing the display orientation does not change the HIGEAR Orientation attribute. Changing the HIGEAR Orientation attribute does not change the display orientation.

When saving an image to a format that supports orientation, ImageGear saves the HIGEAR's Orientation attribute to the file's header, and does not take the display orientation into account. If you'd like to save an image using the current display orientation, copy the orientation from the display settings to HIGEAR.

1.2.4.2.2 Multi-Page Documents

Along with a single-page image handle (HIGEAR), ImageGear provides support for multi-page images. The HMIGEAR handle represents an array of single-page images. You can use ImageGear to:

- Create and delete an internal representation of a multi-page image (HMIGEAR handle)
- Open and associate a multi-page image file with an external file
- Access and manipulate pages within the multi-page image
- Manipulate pages in the external image file, such as loading, saving, swapping, and deleting pages
- Retrieve information about multi-page images and about associated external files

Please see Working with Multi-Page Documents for more information.

1.2.4.2.3 Accessing Image Pixels

ImageGear is equipped with several functions that let you get and set the values of individual pixels, rows or columns of pixels, and rectangular groups of pixels. This family of functions is referred to as the "pixel access" functions.

All functions include the acronym "DIB" in their names. Every pixel access function is part of a "_get()/_set()" pair of functions. In other words, for each pixel access type, you can obtain the value(s) of a pixel or group of pixels, and set the value(s) of a pixel or group of pixels.

To obtain descriptions of each pixel access function and view additional sample code, refer to the Core Component API Function Reference.

This section provides information about the following:

- Pixel Access Modes
- Allocating Space for ImageGear Pixel Access
- Getting and Setting Individual Pixels
- Getting and Setting Linear Groups of Pixels
- Getting and Setting a Rectangular Area of Pixels
- Filling DIB Area

1.2.4.2.3.1 Pixel Access Modes

Pixel access functions have two modes of operation: legacy (prior to ImageGear v14.5) and new (ImageGear v14.5 and newer). The default mode is legacy, in which these functions behave the same way they did before v14.5. So if you have existing code written for ImageGear v14.4 or earlier that uses pixel access functions, you shouldn't need to update it.

New pixel access mode provides more access to the new storage system. It lets you work directly with higher bit depths, advanced color spaces, and alpha/extra channel data included with the main channel data.

If you are migrating from legacy mode to new mode, you must be aware of the following differences between these modes:

- For RGB images, color channel order is RGB (in legacy mode, it is BGR)
- DIBs may use bit depths that were not supported by the Legacy mode (i.e., 36-bit, 48-bit RGB)
- Additional color spaces are supported (i.e., LAB, YUV)
- Alpha and extra channel data is included on a per-pixel basis. For example, if you have a 24-bit RGB image with an 8-bit alpha channel, the pixel data will look like RGBA, RGBA, RGBA and so on, where R, G, B, and A are each one byte.
- Pixel packing and raster padding are as follows:

Packing Mode	**Legacy**	**New**
IG_PIXEL_PACKED	1 bit pixels are packed 8 into a byte; 4 bit pixels are packed 2 into a byte; other pixels are not packed. Rasters are padded to DWORD boundary.	1 bit pixels are packed 8 into a byte; other pixels are not packed. • In 32-bit edition of ImageGear, rasters are padded to DWORD boundary. • In 64-bit edition of ImageGear, rasters are padded to QWORD boundary.
IG_PIXEL_UNPACKED	Pixels are not packed; each 1-bit pixel occupies a byte. Rasters are padded to BYTE boundary.	Pixels are not packed; each 1-bit pixel occupies a byte. • In 32-bit edition of ImageGear, rasters are padded to DWORD boundary; • In 64-bit edition of ImageGear, rasters are padded to QWORD boundary.

Note that 1-bit pixels are the only pixels that are packed in the new mode - 8 pixels are stored in each byte. Pixels of any other channel depths are stored using 1, 2, or 4 bytes per channel. If a pixel has more than one channel and use 1 bit per channel, each of its channels will be stored in a separate byte. A channel value with depth of 2-8 bits will be stored in one byte, 9-16 bits in two bytes, and 17-32 bits in four bytes.

To use the new mode, you need to set the DIB.PIX_ACCESS_USE_LEGACY_MODE global control parameter to IG_PIX_ACCESS_MODE_NEW, as shown in the example below.

```
AT_MODE pixAccessMode = IG_PIX_ACCESS_MODE_NEW;
IG_gctrl_item_set("DIB.PIX_ACCESS_USE_LEGACY_MODE", AM_TID_AT_MODE,
        &pixAccessMode, sizeof(pixAccessMode), NULL);
```

If pixel access mode is IG_PIX_ACCESS_MODE_LEGACY, and the image uses a pixel format not supported by the legacy mode, pixel access “…_get” functions convert image pixels into the closest available legacy supported format.

1.2.4.2.3.2 Allocating Space for ImageGear Pixel Access

A common thread for all pixel access _get() functions is that you must provide an array with enough space to accommodate the data that you will receive.

- Use IG_DIB_raster_size_get() to get the size of the array for storing a complete raster of an image.
- Use IG_DIB_pixel_array_size_get to get the size of the array for storing a specified number of pixels.
- Use IG_DIB_pixel_array_size_get to get the size of the array for storing a specified number of pixels from a single row, column, or line.
- Use IG_DIB_area_size_get to get the size of the array for storing a rectangular area of pixels.

1.2.4.2.3.3 Getting and Setting Individual Pixels

There are two pairs of ImageGear functions for getting and setting the value of an individual pixel.

- IG_DIB_pix_get() and IG_DIB_pix_set() get and set a pixel value as a HIGPIXEL object handle.
- IG_DIB_pixel_get() and IG_DIB_pixel_set() get and set a pixel value into / from a byte array. These two functions take into account the current Pixel Access Mode. If pixel access mode is IG_PIX_ACCESS_MODE_LEGACY, and the image uses a pixel format not supported by the legacy mode, the pixel is converted into the closest available legacy supported format. See Pixel Access Modes for more information.

All pixel access functions consider the coordinates 0,0 as the upper left-hand corner of the bitmap data.

1.2.4.2.3.4 Getting and Setting Linear Groups of Pixels

ImageGear lets you get and set the values of linear groups of pixels that run in a horizontal, vertical, or diagonal direction.

IG_DIB_column_get()	Gets the values of a variable-length (vertical) column of pixels. The IG_DIB_column_get() function requires that you set the nX parameter to the horizontal position of the column that you would like to get, and also the first and last row from which you would like to get the pixel values. For this, you set the values of nY1 and nY2. You also give it a pointer to and nLenBytes (size in bytes) of the buffer to which you will store the pixel values. This function will return the actual number of pixel values acquired: lpNumPixels. If the buffer size of nLenBytes is not large enough to accommodate the number of pixels specified by nY1 and nY2, the line of pixels will be truncated, and you will not receive all of the pixels that you specified.
IG_DIB_column_set()	Sets the values of a column of pixels. IG_DIB_column_set() does just the opposite as its _get() counterpart. It takes the pixel values in the buffer and transfers a specified number of them to a HIGEAR image at row nX, and columns nY1 through nY2. You also supply this function with the number of pixel values that you will be setting, where: max # of pixels to set = (nY2 - nY1) + 1
IG_DIB_line_get()	Gets the values of a variable-length line of pixels. Gives you access to the line of pixel values between any two sets of points in the image (i.e., a diagonal line, a horizontal line, or a vertical line). For this reason, it requires that you have set two x coordinates and two y coordinates. If you were to give equal values to either the x pair of coordinates or the y pair of coordinates, you would specify a line that was strictly horizontal or vertical, respectively. These get/set functions require that you give the size in bytes of the buffer and its address.
IG_DIB_line_set()	Sets the values of a line of pixels. Gives you access to the line of pixel values between any two sets of points in the image (i.e., a diagonal line, a horizontal line, or a vertical line). For this reason, it requires that you set two x coordinates and two y coordinates. If you were to give equal values to either the x pair of coordinates or the y pair of coordinates, you would specify a line that was strictly horizontal or vertical, respectively. These get/set functions require that you give the size in bytes of the buffer and its address.
IG_DIB_raster_get()	Gets the values of a full raster line of pixels. Works similarly to the IG_DIB_column_get() and IG_DIB_row_get() functions, except that it will get the values of a full (horizontal) raster line of pixels. This function is quite a bit easier to use than the above functions, however, because you do not need to supply the beginning and ending position of the line. Also, while you must allocate sufficient memory for the data, you do not need to tell ImageGear what number of bytes your buffer contains.
IG_DIB_raster_set()	Sets the values of a full raster line of pixels. Works similarly to the IG_DIB_column_set() and IG_DIB_row_set() functions, except that it will set the values of a full (horizontal) raster line of pixels. This function is quite a bit easier to use than the above functions, however, because you do not need to supply the beginning and ending position of the line. Also, while you must allocate sufficient memory for the data, you do not need to tell ImageGear what number of bytes your buffer contains.
IG_DIB_row_get()	Gets the values of a variable-length (horizontal) row of pixels. This function works exactly like IG_DIB_column_get(), except that it gets the values of a horizontal row of pixels.
IG_DIB_row_set()	Sets the values of a row of pixels. This function works exactly like IG_DIB_column_set(), except that it sets the values of a horizontal row of pixels.

All pixel access functions consider the coordinates 0,0 as the upper left-hand corner of the bitmap data.

Raster and row access API allow packing more than one pixel per byte; to pack more than one pixel per byte, set the nFormat argument to IG_PIXEL_PACKED. For more details, see Pixel Access Modes.

See Also

Allocating Space for ImageGear Pixel Access Functions

1.2.4.2.3.5 Getting and Setting a Rectangular Area of Pixels

Use IG_DIB_area_get() and IG_DIB_area_set() functions for getting and setting the values of a rectangular area of an image.

Use IG_DIB_area_size_get to get the size of the array for storing a rectangular area of pixels.

All pixel access functions consider the coordinates 0,0 as the upper left-hand corner of the bitmap data.

These functions take into account the current pixel access mode (new or legacy). See Pixel Access Modes for more details.

1.2.4.2.3.6 Filling DIB Area

The IG_DIB_flood_fill() function fills an area in the DIB which is surrounded by a border of the specified color.

1.2.4.2.4 Grayscale Look-Up Tables

Grayscale Look Up Tables map a 8…16 bit image to 8 bit grayscale, allowing you to display a specific contrast range of an image, or to apply a non-linear transform to the image pixels for display. Many image processing functions also take the grayscale Look Up Tables into account, and apply processing on the contrast range specified by the grayscale LUT rather on the whole contrast range.

ImageGear provides a set of functions for working with grayscale LUTs. A grayscale LUT object is represented as an opaque handle: HIGLUT. Use the IG_LUT_... group of functions to create, destroy, and access features of grayscale LUTs.

ImageGear allows you to attach grayscale LUTs to images and to image display settings.

An HIGLUT object can have various input and output depths. Both input and output can be signed or unsigned.

The ImageGear Medical component provides a set of API functions that allows you to build grayscale LUTs according to various DICOM display settings.

Example:

```
HIGLUT GrayLUT;
AT_INT index;
// Create a LUT
IG_LUT_create(12, TRUE, 8, FALSE, &GrayLUT);
// Fill the LUT with a linear table, transforming 12-bit signed image to 8-bit
unsigned
for (index = -2048; index<2048; index++)
{
        value = (index + 2048) / 16;
        IG_LUT_item_set(GrayLUT, index, value);
}
IG_image_grayscale_LUT_update_from(hIGear, GrayLUT);
```

See Also:

Displaying Medical Grayscale Images

1.2.4.2.5 Clipboard Operations

The "clipboard" functions provide the ability to cut, copy, and paste to and from the clipboard. With this function group, you can cut or copy all or a portion of an image to the system clipboard, paste the contents of the clipboard into a new HIGEAR image, or even "paste-merge" the contents of the system clipboard into a pre-existing image. You can also check for the existence of data in the clipboard, and check the size of an image in the clipboard. For separate descriptions of each clipboard function and additional sample code, please refer to the Core Component API Function Reference.

This section provides information about the following:

- Copying/Cutting to the Clipboard
- Checking the Contents of the Clipboard
- Pasting an Image from the Clipboard

1.2.4.2.5.1 Copying/Cutting to the Clipboard

You may cut or copy the entire HIGEAR image, or just a specified rectangular portion of the image, to the clipboard. To copy to the clipboard, call the function IG_clipboard_copy() with the image's HIGEAR handle, and the coordinates of the AT_RECT rectangle that you would like to save to the clipboard. Pass NULL as the rectangle's value to if you want to copy the entire image to the clipboard.

To cut to the clipboard, call IG_clipboard_cut(). The only difference in the prototype of these functions is that IG_clipboard_cut() contains an extra argument for specifying what color pixel to use to replace the pixels that are "cut away". This pixel color argument is usually set to black or white.

1.2.4.2.5.2 Checking the Contents of the Clipboard

ImageGear provides two functions for examining the contents of the clipboard:

- IG_clipboard_paste_available_ex() lets you know whether there is an image in the system clipboard. It is recommended that you always call this function before pasting from the clipboard, and also before calling IG_clipboard_dimensions(). This function returns an AT_BOOL value, where TRUE means that there is a paste-able image in the clipboard.
- IG_clipboard_dimensions() returns three values to you: the width of the image (in pixels), the height of the image (in pixels), and the number of bits per pixel of the image on the clipboard. Using these values, you can determine whether or not the image dimensions are appropriate for your purposes.

1.2.4.2.5.3 Pasting an Image from the Clipboard

There are two ImageGear functions for pasting the image from the clipboard:

- IG_clipboard_paste() creates a new HIGEAR image into which it pastes the contents of the clipboard.
- IG_clipboard_paste_merge_ex() pastes the clipboard image into an existing HIGEAR image at the specified position. If the clipboard image's width is greater than the image into which it is being pasted, it will automatically be cropped to fit; the size of the original HIGEAR image will not change.

Before you call IG_clipboard_paste_merge_ex(), you can call the function IG_clipboard_paste_op_set() to specify the kind of arithmetic operation you want to apply to the pixels of the two bitmaps that intersect during the paste-merge. IG_clipboard_paste_op_set() takes an AT_MODE constant (defined in accucnst.h) that has a prefix of IG_ARITH_. The full group of arithmetic constants is listed under the function description for IG_clipboard_paste_op_set(). ImageGear also supplies a companion reading function IG_clipboard_paste_op_get() to read the current setting for the paste-merge arithmetic operation. See Example code below:

```
AT_DIMENSION nWi, nHi;
UINT nBpp;
BOOL bPasteAvail;
AT_ERRCOUNT nErrcount;
HIGEAR hIGear, hIGear2;
AT_RECT rcClipRect;
nErrcount = IG_load_file("picture.bmp", &hIGear);
if (nErrcount == 0)
{
    nErrcount = IG_image_dimensions_get ( hIGear, &nWid, &nHi, &nBpp );*/
    if ( nErrcount == 0 ) /* If valid image dimensions */
    {/* send the bottom half of the image*/
        rcClipRect.top = nHi/2; /* to the clipboard */
        rcClipRect.left = 0;
        rcClipRect.right = nWi - 1;
        rcClipRect.bottom = nHi - 1;
        nErrcount = IG_clipboard_copy (hIGear, &rcClipRect);
    }
if (nErrcount == 0)
{
/*load a second image into which to merge the clipboard contents*/
        nErrcount = IG_load_file("picture2.bmp", &hIGear2);
}
if (nErrcount == 0)
{
    nErrcount = IG_clipboard_paste_available_ex(&bPasteAvail);
    if (bPasteAvail == TRUE)
    {
    /* set the paste-merge arithmetic operation to Img1^Img2 */
    nErrcount = IG_clipboard_paste_op_set(hIGear,
                IG_ARITH_XOR);
    /* merge clipboard's rectangular contents with upper left
     corner at position 0,0         */
        nErrcount = IG_clipboard_paste_merge_ex(hIGear2, 0 , 0);
        }
}
```

1.2.4.2.6 Run Ends Image Storage Format

As of ImageGear v14.5, the Windows DIB format is no longer used for internal storage of images. So 1-bit images are now always stored internally in run ends format (also called "run lengths" format). Previous versions of ImageGear had IG_IP_convert_runs_to_DIB() and IG_IP_convert_DIB_to_runs() functions that converted the internally stored image between run ends and DIB (uncompressed packed) format. These functions are no longer necessary because conversion is performed automatically as needed.

When you read an image's pixel data using a pixel access function such as IG_DIB_raster_get(), ImageGear decompresses the pixel data for that raster and stores it in your buffer. You can specify packed (8 pixels per byte) or unpacked (1 pixel per byte) format. When you write pixel data using a pixel access function such as IG_DIB_raster_set(), ImageGear will compress and store the pixel data in run ends format. It's important to realize that this decompression and compression is the same work that was previously performed in IG_IP_convert_runs_to_DIB() and IG_IP_convert_DIB_to_runs(). However, instead of being performed on the entire image before and after processing, this work is performed on parts of the image during processing.

This section provides the following information:

- Decompressing and Compressing the Entire Image
- Run Ends Format Description
- Accessing Run Ends Data
- Sample Run Ends Code

1.2.4.2.6.1 Decompressing and Compressing the Entire Image

If you want to work with an image directly and avoid using pixel access functions on a per-raster basis, you can decompress the image to your own buffer using IG_DIB_area_get(). Here's an example scenario:

1. Call IG_DIB_area_size_get to get the size of the buffer.
2. Allocate the buffer (i.e., with new or malloc).
3. Call IG_DIB_area_get. ImageGear decompresses the image into your buffer.
4. Read/write uncompressed pixel data directly in your buffer.
5. Call IG_DIB_area_set if updating ImageGear's internal copy of the image is desired. ImageGear compresses the image from your buffer.

1.2.4.2.6.2 Run Ends Format Description

The run ends format is a specialized variant of run length encoding. Run length encoding relies on the fact that certain types of images frequently contain parts where many adjacent pixels share the same color. A description of such an occurrence is known as a run. Typically a run is described as 1) a color, and 2) the number of following pixels that are that color. An image raster (or entire image) can be stored as a collection of runs. For example, an image of this page could be described as "2000 white pixels, 5 black pixels, 15 white pixels, 5 black pixels, 15 white pixels, 5 black pixels, 30 white pixels" and so on.

Since the run ends format only works on 1-bit images, it can take advantage of the fact that there are only two possible colors present in the raster: 0 and 1. Since there are only two possible colors, the color does not need to be stored for each run. It is inferred from the previous run. Also, having only two colors makes it especially likely that long runs of identically colored pixels will occur, as compared to images with more colors present.

The following points characterize the run ends format:

- An image is stored as a collection of rasters encoded in run ends format. Each raster is independent - there is no information shared between rasters. Therefore, consider only a single raster when thinking about the run ends format.
- A run ends raster is stored as an array of run ends. A run end is a value of type AT_RUN which marks the end of a run by storing the horizontal position (X-coordinate) of where the next run begins.
- Run ends are stored in order from left to right.
- It is always assumed that the first run in a raster is white. If it is not, there will be a "null run" at the beginning of the raster which ends at column 0. This is a means of getting the first real (non-zero-length) run to be black.
- The last run end in the raster is always equal to the image width. This value is stored three times to mark the end of the raster.

Here are some examples of rasters that are 8 pixels in width. Each raster is shown first in uncompressed format, then in run ends format as it would be stored in memory on a 32-bit x86 platform. That is, the number 5 is stored in memory as "05 00 00 00".

Example 1

```
11001000
02 00 00 00 // white run until column 2
04 00 00 00 // black run until column 4
05 00 00 00 // white run until column 5
08 00 00 00 // done (remainder is black)
08 00 00 00
08 00 00 00
```

Example 2

```
00000101
00 00 00 00 // *get first run to be black*
05 00 00 00 // black run until column 5
06 00 00 00 // white run until column 6
07 00 00 00 // black run until column 7
08 00 00 00 // done (remainder is white)
08 00 00 00
08 00 00 00
```

1.2.4.2.6.3 Accessing Run Ends Data

There are two ways that you can access run ends data:

- IG_runs_row_get()/IG_runs_row_set() allow you to read and write rows of run ends data.
  - IG_runs_row_get() retrieves a pointer to the run ends data.
  - IG_runs_row_set() updates a row with compressed data from a buffer you supply.

  These functions are the recommended way of accessing run ends data. The format of the data is exactly as described in the previous section.
- IG_image_DIB_raster_pntr_get() is a general purpose function for getting a pointer to pixel data for a given raster. If you use it on a 1bpp image, it will return a pointer to a run ends raster. You can access this raster directly, but be aware of the following:
  - There is an additional AT_RUN value at the beginning of the raster. This value is equal to the total number of AT_RUN values used to store the raster, including this value. For example, for the raster "11001000", this value would be 7.
  - You cannot write data that exceeds the original length of a raster, because ImageGear allocates only enough space to hold the runs for that raster. For this reason, it is safer to use IG_runs_row_set(), which can reallocate if necessary.
  - Run ends rasters are not stored contiguously in memory. You must call IG_image_DIB_raster_pntr_get() for each raster you want to process.

1.2.4.2.6.4 Sample Run Ends Code

The following is a sample function that decompresses a run ends raster into uncompressed unpacked (1 byte per pixel) format. It's designed to work with the data you would get from IG_runs_row_get().

```
// runsToUnpacked: Decompresses a run ends raster to unpacked format.
//   nWidth - width of image in pixels
//   lpRuns - pointer to input buffer containing run ends data
//   lpPixels - pointer to output buffer to receive unpacked pixel data
void runsToUnpacked(AT_DIMENSION nWidth, LPAT_RUN lpRuns, LPAT_PIXEL lpPixels)
{
        // Starting color is white
        AT_PIXEL outputPixColor = 1;
        // Loop through runs
        AT_INT outputPixPos = 0;
        while (1)
        {
                // Find out when the current run ends
                AT_RUN runEnd = *lpRuns++;
                // Fill in pixels for this run
                while (outputPixPos < runEnd)
                        lpPixels[outputPixPos++] = outputPixColor;
                // Have we reached the end?
                if (outputPixPos >= nWidth)
                        break;
                // Switch colors for next run
                outputPixColor = !outputPixColor;
        }
}
```

The following is a more minimalist view of the same function:

```
void runsToUnpacked(AT_DIMENSION w, LPAT_RUN lpRuns, LPAT_PIXEL lpPixels)
{
        AT_PIXEL c = 1;
        AT_INT x = 0;
        while (1)
        {
                AT_RUN r = *lpRuns++;
                while (x < r)
                        lpPixels[x++] = c;
                if (x >= w)
                        break;
                c = !c;
        }
}
```

The following is a more complex function that creates a 90-degree rotated copy of an image. It operates entirely on run ends data without ever decompressing the data. Note that this is only sample code. This does not represent how ImageGear works internally. Also, error handling is omitted.

```
// Returns a 90-degree rotated copy of the source image
HIGEAR rotate90(HIGEAR hImageSrc)
{
        HIGEAR hImageDst = NULL;
        HIGDIBINFO hDIB;
        AT_INT d[1] = { 1 };
        AT_DIMENSION srcWidth, srcHeight, dstWidth, dstHeight;
        AT_PIXPOS x, y;
        // Get info about source image
        IG_image_dimensions_get(hImageSrc, &srcWidth, &srcHeight, NULL);
        // Create destination image
        dstWidth = srcHeight;
        dstHeight = srcWidth;
        IG_DIB_info_create(&hDIB, dstWidth, dstHeight, IG_COLOR_SPACE_ID_I, 1, d);
        IG_DIB_palette_alloc(hDIB);
        IG_image_create(hDIB, &hImageDst);
IG_DIB_info_delete(hDIB);
        AT_RGB rgb = { 255, 255, 255 };
        IG_palette_entry_set(hImageDst, &rgb, 1);
        // Make a list of source raster pointers
        LPAT_RUN *lpSrcRasters = NULL;
        lpSrcRasters = (LPAT_RUN *) malloc(sizeof(LPAT_RUN) * srcHeight);
        // Make a list of current colors for source runs
        LPAT_BYTE lpSrcRunColors = NULL;
        lpSrcRunColors = (LPAT_BYTE) malloc(sizeof(AT_BYTE) * srcHeight);
        // Populate the lists
        for (y = 0; y < srcHeight; y++)
        {
                AT_RUN runCount;
                IG_runs_row_get(hImageSrc, y, &runCount, &lpSrcRasters[y]);
                if (*lpSrcRasters[y])
                        lpSrcRunColors[y] = 1;
                else
                {
                        lpSrcRunColors[y] = 0;
                        lpSrcRasters[y]++;
                }
        }
        // Allocate a raster large enough to store worst-case input data
        LPAT_RUN lpDstRaster = (LPAT_RUN) malloc(sizeof(AT_RUN) * (dstWidth + 4));
        // Loop through output rasters
        for (y = 0; y < dstHeight; y++)
        {
                AT_INT nDstRuns = 0;
                AT_BYTE dstRunColor = 1;
                AT_INT srcRasterIndex = srcHeight - 1;
                // If the first source pixel is black,
// set us up to start with black in the output raster
                if (!lpSrcRunColors[srcRasterIndex])
                {
                        dstRunColor = 0;
                        lpDstRaster[nDstRuns++] = 0;
                }
                // Loop through columns in destination image
                for (x = 0; x < dstWidth; x++)
                {
                        // Check the color of the run in the source raster that
                        // corresponds to the current column in the destination raster.
                        // Is it the same color as the run we're currently constructing?
                        if (lpSrcRunColors[srcRasterIndex] != dstRunColor)
                        {
                                // If not, then we need to store the run we've been making
// in the destination raster.
                                lpDstRaster[nDstRuns++] = x;
                                // Alternate the current destination run color
                                dstRunColor = !dstRunColor;
                        }
                        // See if it's time to move on to the next source run for
// this source raster
                        if (*lpSrcRasters[srcRasterIndex] == y)
                        {
                                lpSrcRasters[srcRasterIndex]++;
                                lpSrcRunColors[srcRasterIndex] =
!lpSrcRunColors[srcRasterIndex];
                        }
                        // Move on to the next source raster (go *up* through the source
image)
                        srcRasterIndex--;
                }
                // Add the three ending runs to the destination raster
                lpDstRaster[nDstRuns++] = dstWidth;
                lpDstRaster[nDstRuns++] = dstWidth;
                lpDstRaster[nDstRuns++] = dstWidth;
                // Store the destination raster!
                IG_runs_row_set(hImageDst, y, nDstRuns, lpDstRaster);
        }
        // Clean up
        free(lpSrcRasters);
        free(lpSrcRunColors);
        free(lpDstRaster);
        return hImageDst;
}
```

1.2.4.2.7 Working with Image Utility Functions

ImageGear's image utility family of functions provides the capabilities to create, import, and export images in either DIB or DDB format, and to obtain information about any image for which you have a HIGEAR handle. You can also obtain information about image files stored on mass storage devices, such as the file format type, compression, width, height, bits per pixel, or number of pages if a multi-page file.

A few special purpose image utility functions tell you whether a HIGEAR variable contains a valid image handle, and whether an image is grayscale. In addition, there are six functions to help you read and write palettes, either whole or one entry at a time, and save them to disk. There's also a function that lets you set special control options that alter the operation of ImageGear's file format read-write filters during file operations.

One additional important image utility function sets an image's "image rectangle", which determines the portion of the image to be displayed, printed, or saved during display, print, and save operations. Refer to Core Component API Reference for detailed calling sequences and further notes on these functions.

This section provides the following information:

- Creating DIBs and DDBs
- Deleting DIBs and DDBs
- Reading and Writing Palettes
- Getting Information about a HIGEAR Image

1.2.4.2.7.1 Creating DIBs and DDBs

This section provides information about how to create DIBs and DDBs.

- To create a new HIGEAR with an empty DIB, use the following code (see IG_image_create_DIB_ex):

  ```
  IG_image_create_DIB_ex ( nWidth, nHeight, nBpp, lCompression, lpDIB = NULL, &hIGear
  );
  ```

  Later, you could use the IG_DIB_...() for direct pixel access functions, or an IG_IP_blend_ ...() function to create an image bitmap. You can use the IG_palette_ ...() functions (described later in this section) to add a palette to the DIB.

- To create a new HIGEAR with a DIB filled by copying an existing DIB, use the same function as demonstrated above, but call it like this:

  ```
  IG_image_create_DIB_ex ( 0, 0, 0, 0, lpDIB, &hIGear );
  ```

  In this example, lpDIB is a pointer to the existing DIB to copy, and the first four arguments are ignored. Width, height, bits per pixel, and "compression" will be copied from the existing DIB, along with its image bitmap and palette.

- To give a HIGEAR handle to an existing DIB, use the import function:

  ```
  IG_image_DIB_import ( lpDIB, &hIGear );
  ```

  This call returns you to the HIGEAR handle assigned to your DIB.

- To create a HIGEAR whose image is a copy of the image in an existing DDB, use IG_dspl_DDB_import:

  ```
  IG_dspl_DDB_import ( hBitmap, NULL, &hIGear);
  ```

  In this call, you provide the DDB's HBITMAP handle that is actually an CGImageRef object. ImageGear creates a DIB for you and returns the new DIB's HIGEAR handle.

- To create a DDB whose image is a copy of the image in an existing HIGEAR, use IG_dspl_DDB_create:

  ```
  IG_dspl_DDB_create( hIGear, IG_GRP_DEFAULT, hDC, nWidth, nHeight, TRUE, &hBitmap,
  NULL );
  ```

  Provide the HIGEAR handle and the width and height for your DDB as well as the addresses of an HBITMAP (CGImageRef) variable to receive the DDB. Because this function belongs to the display group, there should be a group identifier, nGripID, to specify where to get the options needed to complete this operation. Pixel format of DDB being created is 32-bit RGB.

1.2.4.2.7.2 Deleting DIBs and DDBs

This section provides information about how to delete DIBs and DDBs.

- To delete the HIGEAR, but keep the DIB in existence (and obtain its address):

  ```
  IG_image_DIB_export( hIGear, lpDIB, DIBSize, &Options);
  ```

  The DIB's address is returned in your LPAT_DIB variable lpDIB.

- To delete the HIGEAR and the DIB, but produce a copy of the image in DDB format, use IG_dspl_DDB_create:

  ```
  IG_dspl_DDB_create( hIGear, IG_GRP_DEFAULT, hDC, nWidth, nHeight, TRUE, &hBitmap,
  NULL )
  ```

  Provide the HIGEAR handle of the image to delete, and provide the addresses of the HBITMAP (CGImageRef) variables to receive the handle for the created DDB.

- To delete a HIGEAR, including the DIB, when you are entirely done using it, use the following call to IG_image_delete:

  ```
  IG_image_delete ( hIGear );
  ```

Note that in all the above cases, ImageGear will not release memory that it did not allocate. For example, assume your application has allocated memory, created a DIB, and subsequently used an IG_image_DIB_import() call to give this DIB a HIGEAR handle. In this case, a later call to IG_image_delete() will delete the HIGEAR structure but will not free the DIB's memory. The owner of the DIB's memory (in this case, your application) would issue a free() call to free this memory.

To delete a DDB being created with IG_dspl_DDB_create, use a CGImageRelease(hBitmap) call.

1.2.4.2.7.3 Reading and Writing Palettes

There are six image utility functions that read and write palettes in memory, or load and save them between memory and disk. To transfer a palette between your own memory area and a HIGEAR image's DIB (that is, to get or set the DIB palette), use IG_palette_get and IG_palette_set:

```
AT_RGBQUAD palette[256];
IG_palette_get ( hIGear, palette );
IG_palette_set ( hIGear, palette );
```

In the above calls, lpPalette should point to the first of an array of AT_RGBQUAD structures, one structure per palette entry.

If instead you want to move just one entry to or from the DIB palette, use IG_palette_entry_get and IG_palette_entry_set:

```
AT_RGB    rgbPaletteColor;
IG_palette_entry_get ( hIGear, &rgbPaletteColor, nIndex );
IG_palette_entry_set ( hIGear, &rgbPaletteColor, nIndex );
```

In the single entry calls, you supply a pointer to a single structure of type AT_RGB. The third argument has a value between 0 and 255, specifying which palette entry to get or set. (Remember that an AT_RGBQUAD structure consists of 4 bytes ordered Blue-Green-Red-Unused(0), while an AT_RGB struct consists of 3 bytes ordered Blue-Green-Red.)

To load and save palettes between memory and disk, use the functions IG_palette_load and IG_palette_save:

```
IG_palette_load ("filename", palette, nEntries, bOrder, lpFileType);
IG_palette_save ("filename", palette, nEntries, lpFileType);
```

See the descriptions of the above functions in Core Component API Reference for details on specifying the arguments.

1.2.4.2.7.4 Getting Information about a HIGEAR Image

This section provides instructions on getting information for a HIGEAR.

- To find out if a HIGEAR variable currently holds a valid handle, call IG_image_is_valid:

```
if ( IG_image_is_valid(hIGear) )  { ... }
```

- Similarly, to find out if it is a grayscale image, call IG_image_is_gray:

```
if ( IG_image_is_gray(hIGear, &bItsGray) )  { ... }
```

- To obtain the width, height, and bits per pixel of an image, or the DIB compression type, use IG_image_dimensions_get or IG_image_compression_type_get (respectively):

```
AT_DIMENSION nWidth, nHeight;
UINT nBpp;IG_image_dimensions_get ( hIGear, &nWidth, &nHeight, &nBpp );
DWORD nCompression;IG_image_compression_type_get ( hIGear, &nCompression );
```

> Be careful to declare the types as AT_DIMENSION and DWORD where shown above (rather than INT or UINT). On some development platforms, AT_DIMENSION and DWORD are not the same size as INT.

- To get the position and size of an image's image rectangle, use IG_dspl_layout_get() API.

> Here also be careful to use the correct type: AT_RECT, not Windows structure type RECT, whose fields may be a different length.

- To set the image rectangle, use the function IG_dspl_layout_set(). Examples are provided in the sections Displaying Images and Saving Images.

If you need to access a DIB directly, refer to IG_image_DIB_palette_pntr_get().

1.2.4.2.8 Working with Gigabyte-Sized Images

When you load an image into ImageGear image handle, or create one, its pixel data is stored in the computer's random access memory (RAM) by default. As the physical memory usage grows, the system swaps less used blocks of memory from running applications to the system Page file. If an application tries to allocate a block of memory comparable with the computer's RAM size, the system has to push its own resources to the Page file. This makes the system extremely unresponsive. If an application requests more memory than (size of the RAM + the page file size – amount of memory used by the system), such request cannot be fulfilled, and the allocation fails.

ImageGear allows working with such large images by allocating memory for the DIB via a memory mapped file. On a 64-bit operating system, this allows allocating images nearly as large as the amount of free disk space on the computer, without overloading the RAM and affecting the system responsiveness. On a 32-bit OS, maximum total size of DIBs allocated simultaneously in several processes cannot exceed 3…3.5 Gb, and the size of all DIBs allocated in one process cannot be greater than 2 Gb; however, using memory mapped files still makes working with large images much more convenient.

If the images are not big, or there is plenty of free RAM, keeping image pixels in the RAM provides better performance than using the memory mapped files. However, when image size is comparable to RAM size, or is greater, memory mapped file usage provides much better performance than storing the image in memory.

ImageGear does not use memory mapped files for 1-bit images. However, ImageGear uses Run Ends compression for storing them, so they rarely occupy large amounts of memory.

This section provides the following information:

- Quick Start
- How to Configure
- Accessing Pixels of a Gigabyte-Sized Image
- Reading and Writing Gigabyte-Sized Image Files

1.2.4.2.8.1 Quick Start

By default, parameters that improve processing of large images are disabled in ImageGear. Follow these steps to try ImageGear's enhanced support for large images:

1. Run the Image Processing sample.
2. Go to **Main** menu > **Settings** > **Parameters...**
3. Select "DIB.FILE_MAPPING.THRESHOLD" in the list of parameters and set its value to 500. This enables memory mapping file storage for DIBs that have uncompressed size of 500 Mb and more.
4. Click **Apply** and close the dialog.

The sample is now ready to work with gigabyte-sized images.

> If you try loading a gigabyte-sized image into ImageGear without the file mapping being enabled, the system may become unresponsive because of excessive RAM usage by ImageGear.

For additional convenience you can also enable the progress bar. Note though that ImageGear display operations do not trigger the progress bar.

- Go to **Main** menu > **Settings** > **Progress Bar**.

If you are not particularly interested in fine display for large images, you can turn the display interpolation off (after the image has been loaded). This will result in much faster image display.

- Go to **Main** menu > **View** > **Anti-aliasing**, and uncheck the **Color Antialiasing** and **Use Resampling** check boxes.

1.2.4.2.8.2 How to Configure

ImageGear handles all of the memory mapped file operations internally, except for a specific case discussed in the next section. The application only needs to set a few parameters to enable the usage of memory mapped files and adjust it to its needs.

The usage of memory mapped files is controlled by three global control parameters:

- **"DIB.FILE_MAPPING.THRESHOLD"**. Specifies minimum DIB size, in megabytes, for which the memory mapped file shall be used. DIBs that are smaller than this threshold are allocated in physical memory. By default, this parameter is set to 0, which means that the use of memory mapped files is disabled. 1-bit images are always allocated in the physical memory and are not affected by this parameter.
- **"DIB.FILE_MAPPING.PATH"**. Specifies the path to a folder where memory mapped files will be stored. Memory mapped files are temporary files that are created upon DIB creation and deleted upon its deletion. By default, this parameter is set to an empty string, which means that ImageGear will use the system temporary folder for memory mapped files. This parameter is only used when the "DIB.FILE_MAPPING.THRESHOLD" parameter value is greater than zero. Does not affect 1-bit images.

For best performance, use a separate SSD drive or hard drive for storing memory mapped files, and make sure that the operating system and other applications do not use this drive.

- **"DIB.FILE_MAPPING.FLUSH_SIZE"**. Specifies the maximum size of a memory block in a DIB that can be processed without flushing the memory mapped file. The default value is 200 Mb, which shall be efficient on typical systems that have 4 Gb of RAM. Greater values may improve performance on systems that have a larger amount of RAM. However, making this value too big (comparable to RAM size) will impact the system responsiveness and may lead to allocation failure for large DIBs. This parameter is only used when the "DIB.FILE_MAPPING.THRESHOLD" parameter is greater than zero. Does not affect 1-bit images.

All three parameters DIB.FILE_MAPPING.THRESHOLD, DIB.FILE_MAPPING.PATH and DIB.FILE_MAPPING.FLUSH_SIZE are taken into account at the time of the DIB creation. After the DIB has been created, changing these parameters will not have an effect on the storage of this DIB, or flush frequency during its processing. However, if some operation replaces the DIB in a HIGEAR (e.g., Resize, Rotate), a new DIB will be created according to the current DIB.FILE_MAPPING.THRESHOLD and DIB.FILE_MAPPING.PATH values, and will then be processed according to the DIB.FILE_MAPPING.FLUSH_SIZE value at the time of the DIB replacement.

The following example tells ImageGear to use memory mapped files for images whose pixel data size is equal to or greater than 500 MB:

```
// Memory mapping will be used for DIBs with sizes equal to or greater than 500 Mb.
AT_INT fileMappingThreshold = 500;
IG_gctrl_item_set("DIB.FILE_MAPPING.THRESHOLD", AM_TID_INT, &fileMappingThreshold,
sizeof(fileMappingThreshold), "");
```

The following example obtains the current value of the DIB.FILE_MAPPING.THRESHOLD parameter:

```
// Get memory mapping threshold
AT_INT fileMappingThreshold;
IG_gctrl_item_get("DIB.FILE_MAPPING.THRESHOLD", NULL, (LPVOID)&fileMappingThreshold,
sizeof(fileMappingThreshold), NULL, NULL, 0, NULL);
```

The following example tells ImageGear to create temporary memory-mapped files in the current directory:

```
// Use current directory for the memory mapped files
char* szMemoryMappingPath = ".";
IG_gctrl_item_set("DIB.FILE_MAPPING.PATH", AM_TID_MAKELP(AM_TID_CHAR),
szMemoryMappingPath, (DWORD)strlen(szMemoryMappingPath) + 1, "");
```

The following example obtains the directory used for storing memory mapped files:

```
// Get path for memory mapped files
char szMemoryMappingPath[_MAX_PATH];
IG_gctrl_item_get("DIB.FILE_MAPPING.PATH", NULL, (LPVOID)&szMemoryMappingPath,
sizeof(szMemoryMappingPath) - 1, NULL, NULL, 0, NULL);
```

The following example tells ImageGear to set the flush size to 100 Mb:

```
// Memory mapping flush size will be equal to 200 Mb.
AT_INT fileMappingFlushSize = 200;
IG_gctrl_item_set("DIB.FILE_MAPPING.FLUSH_SIZE", AM_TID_INT, &fileMappingFlushSize,
sizeof(fileMappingFlushSize), "");
```

The following example obtains the current flush size:

```
// Get memory mapping flush size
AT_INT fileMappingFlushSize;
IG_gctrl_item_get("DIB.FILE_MAPPING.FLUSH_SIZE", NULL, (LPVOID)&fileMappingFlushSize,
sizeof(fileMappingFlushSize), NULL, NULL, 0, NULL);
```

1.2.4.2.8.3 Accessing Pixels of a Gigabyte-Sized Image

ImageGear manages the use of memory mapped files internally. In most cases, the application does not need any additional code for working with memory mapped images, except for setting the global control parameters. However, if an application accesses individual pixels or rasters of a large image, or accesses pixel data directly by the image or raster pointer, ImageGear does not know when to flush the image's memory mapped file. In this case the application shall flush the memory mapped file explicitly, using IG_DIB_flush.

```
AT_DIMENSION nRasterSize;
AT_INT nRasterCountToFlush;
AT_INT i;
AT_PIXEL* nBuffer;
AT_DIMENSION nImageHeight;
AT_ERRCOUNT nErrCount;
HIGEAR hIGear;

// Load the image
nErrCount = IG_load_file("picture.tif", &hIGear);
if( nErrCount == 0 )
{
    IG_image_dimensions_get(hIGear, NULL, &nImageHeight, NULL);
    IG_DIB_raster_size_get(hIGear, IG_PIXEL_UNPACKED, &nRasterSize);
    nRasterCountToFlush = 200 * 1024 * 1024 / nRasterSize;
    nBuffer = new AT_PIXEL[nRasterSize];

    for(i = 0; i < nImageHeight; i ++)
    {
        // Get image raster
        IG_DIB_raster_get(hIGear, i, nBuffer, IG_PIXEL_UNPACKED);

        // Process the raster
        // ...

        if((i + 1) % nRasterCountToFlush == 0)
        {
            // We have accessed about 200 MB of sequential memory.
            // Flush the memory-mapped file associated with the image.
            IG_DIB_flush(hIGear);
        }
    }
    // Flush the memory-mapped file at the end of pixel access.
    IG_DIB_flush(hIGear);

    delete[] nBuffer;
    IG_image_delete(hIGear);
}
```

A similar situation occurs when the application accesses image areas. Since areas can be small, even one automatic flush per area access operation can be too much, and can significantly degrade performance. ImageGear does not use automatic flushing in area access operations. Instead, the application shall flush the DIB after accessing one or several areas. If an area to be accessed is very large (hundreds of megabytes or more), we recommend splitting it into smaller areas and processing it sub-area by sub-area.

In order to minimize the amount of flushing, try to prefer row-wise order of processing, as opposed to column-wise. For example, if you need to access an image area consisting of 10 sub-areas vertically and 10 sub-areas horizontally, do it as follows:

1. Access the first row of sub-areas
2. Flush the image
3. Access the second row of sub-areas
4. Etc.

1.2.4.2.8.4 Reading and Writing Gigabyte-Sized Image Files

The two most important factors in image file formats that affect their ability to support gigabyte-sized images are as follows:

- Maximum allowed image dimensions. This usually depends on the integer format used for storing image dimensions.
- File size limitation. This usually depends on the integer format used for storing lengths or offsets to various data in the file. If a file is stored compressed, image size after decompression may be greater than the maximum supported file size.
  - Some file formats have limitations on blocks (chunks, strips) of pixel data, but allow multiple such blocks to exist in a file, and thus avoid the limitation on the file size.

Although some file formats allow storing gigabyte-sized images, particular software may have difficulties with reading or writing them.

The table below lists some of the popular file formats and their capabilities for storing gigabyte-sized images.

Image Format	**Max Available Image Dimensions (width x height, pixels)**	**Max Image Size, When Uncompressed, Approximately (for a 24-bit RGB image)**	**File Size Limit, Approximately**
JPEG, EXIF JPEG	65535 x 65535	12 Gb	None
TIFF, EXIF TIFF	2^32-1 x 2^32-1	3 * 2^24 Tb	4 Gb [1]
JP2,JPX	2^32-1 x 2^32-1	3 * 2^24 Tb	None
PSB	300 000 x 300 000	250Gb	None
PSD	30 000 x 30 000	2,5Gb	4 Gb
BMP	2^31-1x 2^31-1	3 * 2^20Tb	4 Gb
PNG	2^32-1 x 2^32 - 1	3* 2^24Tb	None
DICOM	65535 x 65535	12 Gb	2 Gb
PBM / PGM / PPM / PNM	None	None	None
TGA	65535 x 65535	12 Gb	4 Gb

[1] TIFF format uses 32-bit unsigned integers to store data offsets and sizes. As a result, a strip of pixel data in a TIFF image cannot be stored at an offset greater than 4 Gb, and its size formally cannot be greater than 4 Gb. Thus, the size of the largest compliant TIFF image can be a bit less than 8 Gb. This assumes that two strips of nearly 4 Gb size are used.

ImageGear supports the reading and writing of single-page, single-strip, single-tiled uncompressed TIFF images where strip byte counts are greater than 4 Gb. If the size of a strip exceeds 4 Gb, ImageGear writes 0 to the StripBytes tag. The reader can calculate strip size from image dimensions in such a case.
Note, though, that such files are formally incompliant and may not be supported by other readers.

When writing a gigabyte-sized TIFF image, make sure to keep the “IMAGE_BEFORE_IFD” TIFF control parameter set to its default value of FALSE.

1.2.4.3 Loading and Saving Images

This section provides information about the following:

1.2.4.3.1 Loading Images

The IG_load_...() functions provide the means to bring images from image files into ImageGear's sphere of influence. The image files may be on a mass storage device such as a disk, or they may already be in memory.

When you load an image using an IG_load_...() function, ImageGear provides a handle of ImageGear type HIGEAR. Having HIGEAR handles for your images allows your application to perform the entire range of ImageGear's imaging operations.

Alternately, you can import images that exist as plain bitmaps, DIBs, or DDBs in your application. You can scan images from elsewhere directly into ImageGear. You can locate and extract ASCII or graphics images from multi-page (multi-image) files, using ImageGear's GUI browse and other capabilities.

IG_load_ ...() functions create a DIB in memory, and transfer the bitmap and other pertinent information (such as the image's color palette, if one is associated with it; and header information such as width, height, and bits per pixel) into this DIB. These functions do not display the image unless the word "display" is included in the function name, such as in IG_load_file_display().

For information about loading CMYK images, see Color Management.

Here are a few examples that demonstrate how to call the functions in this group. Refer to "Core Component API Function Reference" for more information on the IG_load_ ...() functions.

Example 1:

```
#include "gear.h"
HIGEAR         hIGear;         /* HIGEAR handle returned   */
AT_ERRCOUNT    nErrcount;      /* Count of errors reported */
nErrcount = IG_load_file ("picture.bmp", &hIGear);
```

The example code above loads the file "picture.bmp" from the current directory, creating a DIB in memory, and creates a unique ImageGear handle for it, returning this handle to you in hIGear.

Example 2:

```
HIGEAR         hIGear;         /* handle returned by ImageGear  */
char *         lpWhereFile;     /* ptr to image file in mem      */
DWORD          dwWholeSize;   /* size of image file in mem     */
UINT           nPageNum;                /* will be 0 for this call     */
AT_ERRCOUNT    nErrcount;   /* to test for errors                */
nPageNum      = 0;         /* not a multi-page file          */
lpWhereFile   = ...;     /* where mem image file begins    */
dwWholeSize   = ...;      /* size of whole mem image        */
nErrcount = IG_load_mem (lpWhereFile, dwWholeSize,nPageNum, 0, &hIGear);
```

IG_load_mem() is used to load an image from memory. The image in memory must be in the same format as if the image were on disk. For example, the image must contain an appropriate header, the bitmap data, and if necessary, a palette. This in-memory file must be of a format recognized by ImageGear (see "File Format Reference").

In the above call, dwWholeSize must be the size of the entire memory image, not just of the bitmap. lpWhereFile must point to the first byte of this whole image. And nPageNum, if not zero, specifies the page number to load if loading from a multi-page (multiple image) file. Please note that the first page of a multi-page file is page number 1, not page number 0. The function IG_load_mem() creates a DIB and loads into it the image specified, and returns in hIGear the new HIGEAR image handle by which you will refer to this image in subsequent calls to ImageGear functions.

Example 3:

```
HIGEAR         hIGear;       /* handle ret'd by IG_load_FD    */
INT            fd;         /* File Descriptor handle         */
LONG           lOffset;      /* offset to image in file       */
UINT           nPageNum;   /* will be 0 for this call        */
```

```
AT_ERRCOUNT nErrcount;    /* to test for errors          */
fd = _lopen ( "picture.bmp", OF_READ );    /* open file     */
nPagenum   = 0;          /* not a multi-page file      */
lOffset    = 0;          /* access file from start     */
/* Load image, and obtain its ImageGear handle: */
nErrcount = IG_load_FD (fd, lOffset, nPageNum, 0, &hIGear);
```

Names of some IG_load_ ...() functions contain the letters "FD", for example IG_load_FD_CB(). These functions access the file by its File Descriptor handle, which is an integer value returned to you when you open the file using certain Windows functions. You may use these IG_load_FD...() functions to load an image from a file, if the file is already open and your application has its File Descriptor handle.

The example above shows ImageGear IG_load_FD...() loading from a file that has been already opened by means of the Windows function _lopen() or other file I/O function that returns a File Descriptor handle.

If nErrcount is zero, a DIB is created, the image is loaded, and your HIGEAR object, &hIGear, contains its HIGEAR handle.

See Also:

Detecting Image File Format

1.2.4.3.1.1 Detecting Image File Format

You can detect the format of an image file before and after loading it in the HIGEAR handle using functions IG_fltr_detect_...():

- IG_fltr_detect_FD(INT fd, LONG lOffset, LPATE_MODE lFileType) detects the format by scanning its File Descriptor fd starting from the lOffset position in the file.
- IG_fltr_detect_file(const LPSTR lpszFileName, LPATE_MODE lFileType) determines the format of the image located in the specified file by given filename.
- IG_fltr_detect_mem(void FAR lpImage, DWORD dwSize, LPATE_MODE lFileType) determines the format of image located in memory buffer.

All three functions return the type of image format as IG_FORMAT_ constant, as delineated in the accucnst.h file.

The code fragment below allows you to sort image files by their formats and then choose how to use them:

```
AT_ERRCOUNT nErrCount = IGE_SUCCESS;/* will hold returned error count */
AT_MODE nFormatID;
...
nErrCount = IG_fltr_detect_file( "image.tiff", &nFormatID );
if( nFormatID==IG_FORMAT_TIF )
{
...
}
```

For better performance or for some other purposes, use the special functions IG_fltr_load_file_format() and IG_fltr_pagecount_file_format() if the image format is known. Those functions accept the first parameter as the format ID and do not perform detect operations like IG_fltr_load_file() does, but immediately start to operate with the data, assuming it from the specified format. If the data is invalid or the image format is different, then an error is placed in the error stack.

1.2.4.3.2 Saving Images

The IG_save_...() family of functions is complementary to the IG_load_...() functions. The IG_save_...() functions allow you to save images to disk files or to memory, convert files from one file format to another, and to append or insert images as pages to a multi-page file. All IG_save_...() functions have the lFormatType parameter in common. This allows you to choose which ImageGear-supported file format and compression type (where applicable) to which to save. Whereas some functions, such as IG_info_get_ex(), have separate parameters for the format and compression types, the saving functions have one parameter that covers both.

This section provides information about the following:

- The Image Rectangle
- Saving Images to a Disk File
- Saving to a Disk File Using a File Descriptor Handle
- Saving an Image to Memory
- Converting Images from One File Format to Another

1.2.4.3.2.1 Saving Images to a Disk File

```
IG_save_file(HIGEAR hIGear, lpszFileName, lFormatType)
```

IG_save_file is the function normally used to save a HIGEAR image to disk. The name of the file to which to save is specified with the second argument, and the format type and compression type (if applicable) in which to save it is specified in the third. If the file format being used supports more than one compression type, more than one constant will be available. The BMP format, for example, is provided with two values for lformatType: IG_SAVE_BMP_UNCOMP and IG_SAVE_BMP_RLE.

The file accucnst.h defines the constants to which lFormatType may be set. These constants are also listed at the end of File Format Reference.

If the file already exists, and the format supports multiple pages, IG_save_file() will append the new image(s) to the file. If the file already exists and it is a single image format type, the file will be overwritten.

The following code example demonstrates the use of IG_save_file():

```
HIGEAR hIGear;              /* handle ret'd by IG_load_...     */
AT_ERRCOUNT nErrcount;      /* # of errors on stack           */
AT_LMODE lFormatType;       /* format type to save to         */
/*set format and compression to an AT_LMODE constant       */
lFormatType = IG_SAVE_TIF_UNCOMP;
nErrcount = IG_save_file(hIGear, "picture.tif", lFormatType);
```

If you do not know the file format, you may set lFormatType to IG_SAVE_UNKNOWN. ImageGear will check the file's extension and save the image accordingly. If you set lFormatType to a value other than IG_SAVE_UNKNOWN, ImageGear assumes that you are specifying the correct type and saves the image accordingly.

Before selecting a file format to which to save, you can reference the section entitled ImageGear Supported Bit Depths to ensure that you save to a format that supports the same bit depth as the original image.

1.2.4.3.2.2 Saving to a Disk File Using a File Descriptor Handle

You can use the IG_save_FD() function to save a HIGEAR image to a file that has already been opened and for which you have a File Descriptor handle (for example, your application may have opened the file using the Windows function _lopen()). IG_save_FD() will permit you to insert an image into a multi-page file (instead of merely appending it, as IG_save_file() would do.)

IG_save_FD() is called with five arguments, of which the HIGEAR image handle and lFormatType have the same meaning as in a call to IG_save_file(). The other parameters are the File Descriptor handle, fd obtained by the Windows function call that opened the file, a reserved argument nReserved, which should always be set to 0, and nPageNum, which allows you to specify the page in which to insert the image. If you want to append image(s) to a file, you can set nPageNum to IG_APPEND_PAGE.

The following example shows IG_save_FD() being used to insert an image as page 3 into an existing a multi-page file:

```
HIGEAR        hIGear;        /* ImageGear handle             */
INT        d;        /* File Descriptor handle    */
UINT        nPageNum,        /* page # to insert image as    */
        nReserved;        /* for future expansion    */
AT_ERRCOUNT nErrcount;                /* # of errors on stack    */
AT_LMODE        lFormatType;        /* type of format to save to            */
/ *Windows call to open file with write privileges                    */
fd = _lopen("picture.tif", WRITE);
nPageNum = 3;            /* save this image as page #        */
nReserved = 0;            * always set to 0 for now        */
lFormatType = IG_SAVE_TIF_JPG; /* format is TIFF-JPEG                    */
/* save the HIGEAR image as page 3 of file whose descriptor is fd:*/
nErrcount = IG_save_FD(hIGear, fd, nPageNum, nReserved,lFormatType,);
```

If the file "picture.tif" has five pages, the new image will be inserted into the position of the third page, and what was formerly page three will now be page four, and so on. Remember that ImageGear treats the first page of a multi-page file as page 1, not 0. Setting the nPageNum parameter to 0 will append your image(s) to the file. If you set nFormatType to a type that doesn't support multiple pages, then the file will be overwritten with a single image.

Inserting pages to a multi-page file does not physically rearrange the image in the file. Rather, the offset table is adjusted to hold the new address(es) of the inserted page(s), and the new order of the image within the file.

lFormatType should be set to one of the IG_SAVE_ constants, which also include the compression type. Note that TIFF has more than one compression type, and that there are different constants to represent TIFF with each type of compression. If you call IG_save_FD() for a multiple-page file, you must supply the format type (from accucnst.h) for the image to be appended to the file. If you do not know the format of the destination file, first call IG_info_get_FD_ex() and use the format information returned to decide which format to use.

The following pseudo-code example demonstrates how to write a multi-page image using IG_save_FD():

```
lFormat = IG_SAVE_TIF_UNCOMP;
fd = _lopen (szFile,OF_READWRITE);
for (i = 1; i<=nPages; i++)
{
/*Seek to the beginning of the file before writing each page ImageGear will
automatically find the correct file position of the new page */
    seek(fd, 0, SEEK_SET );
    /*Write each next page                    */
    nErrcount = IG_save_FD(hIGear, fd, i, 0, lFormat);
}
close(fd);
```

When you call OS SEEK or an equivalent function to set the file pointer, the stream needs to be positioned at the FIRST byte of a multi-page image. This is the only way in which a filter can recognize that the image is multi-page.

A file should be opened with both READ and WRITE access.

1.2.4.3.2.3 Saving an Image to Memory

ImageGear allows you to save images to memory in the same way as you save images to a disk file. See Using Format Filters API for Image Saving / Saving an Image to Memory section for details.

1.2.4.3.2.4 Converting Images from One File Format to Another

Converting a file from one file format to another is simply a matter of loading a file from disk using IG_load_file() or other load function, and then saving it with a new format type. Be sure, however, that the bit depth of the original file can be supported by the new format type. To help you with this, we have provided you with a Format Bit Depths table in the section entitled ImageGear Supported Bit Depths.

The following is an example in which a Windows Bitmap file (BMP) is converted to a TIFF file (.TIF):

```
HIGEAR      hIGear;         /* HIGEAR handle returned        */
AT_ERRCOUNT nErrcount;      /* # of errors on stack          */
AT_LMODE    lFormatType;    /* format to save to             */
/* Set name of BMP file to load and format to save to        */
lFormatType = IG_SAVE_TIF_UNCOMP;
/* Load picture.bmp, obtaining ImageGear handle of image      */
nErrcount = IG_load_file("picture.bmp", &hIGear);
if (nErrcount == 0 )           /* If successful:              */
{
    /* save image as a TIFF file:                             */
    nErrcount = IG_save_file(hIGear, "picture.tif", lFormatType );
}
```

If the file named "picture.tif" already exists, the image will be appended to "picture.tif" because TIFF supports multiple images in a single file.

For more information on the ImageGear IG_save_...() functions, see Core Component API Function Reference.

In order to save an image using ImageGear, it must have a HIGEAR handle. If your image is in DIB format, use: IG_image_DIB_import() or IG_image_create_DIB_ex() to assign it a HIGEAR handle. If the image is a DDB, use IG_dspl_DDB_import().

1.2.4.3.2.5 The Image Rectangle

In ImageGear, an "image rectangle" defines the coordinates for the area of the bitmap that will be saved or displayed. The image rectangle represents a rectangular array of pixels from the bitmap that can include the whole image or any rectangular portion of the image.

You can leave the image rectangle set to its default, which includes the whole image, or you can specify a portion of the image by setting the image rectangle with IG_dspl_layout_set(). A call to this function is included in the example code in Saving an Image to Memory. You can also find a detailed description and example of this function's use in the Core Component API Function Reference.

For a more detailed description of the Image Rectangle and other ImageGear display rectangles, see Geometric Transformations.

1.2.4.3.2.6 Using Format Filters API for Image Saving

The ImageGear filters API provides three saving functions that make your application much faster and more flexible when working with image saving:

- IG_fltr_save_file(HIGEAR hIGear, const LPSTR lpszFileName, AT_LMODE lFormatType, UINT nPageNumber, AT_BOOL bOverwrite)
  This function works similarly to the IG_save_file() function, but has additional arguments that provide additional functionality when saving the image in a multi-page file. The nPageNumberargument allows you to specify the number of the page in an already existing multi-page file where you want the saved page to be placed. The last argument bOverwrite allows you to determine the mode of how to work with multi-page image files. The TRUE value completely overwrites the file and places a single page there, but a FALSE value means that the existing file will be expanded with one additional page specified by the nPageNumber parameter.
- IG_fltr_compressionlist_get(LPAT_DIB lpDIB, AT_MODE nFormatID, LPAT_MODE lpComprList, UINT nCListSize, LPUINT lpnCListCount)
  This function allows you to get information about all compressions (as IG_COMPRESSION_constants returned in lpComprListlist) that are available when saving the image to the file format specified by the nFormatID(as IG_FORMAT_constant) parameters. The first parameter allows you to specify information about the image to be saved. If this parameter is NULL, then the function returns all available compressions, otherwise it returns compressions that are applicable to a given image. For example, G3/4 compressions are only applicable for bi-tonal images, but JPEG compression is only applicable for color images with 8 bits or more per pixel.
- IG_fltr_savelist_get(LPAT_DIB lpDIB, LPAT_MODE lpnFilterList, UINT nFListSize, LPAT_LMODE lpSaveList, UINT nSListSize, LPUINT lpnSListCount)
  This function allows you to prepare the list of applicable compressions, not for a single format but for a list of formats. It provides the ability to quickly build a list of applicable parameters that can be used, for instance, for the third (lFormatType) parameter of IG_fltr_save_file(). The main returned value lpSaveList contains a list of combined values (nFormat | (nCompression<<16)) = (IG_FORMAT_ | IG_COMPRESSION_<<16) available for a given image to save. You can get values of nFormat from lpnFilterList provided through the second parameter. For instance, if you specified lpnFilterList as IG_FORMAT_TIF, then the returned lpSaveList will be looked at as a list of (IG_FORMAT_TIF | IG_COMPRESSION_) values where IG_COMPRESSION_ is a list of all the compressions available for the TIFF image with the current lpDIB. If lpnFilterList is NULL then this function uses all available formats registered in ImageGear and returns the list with all currently available formats and compressions.

You can process the IG_fltr_compressionlist_get() and/or IG_fltr_savelist_get() functions before using IG_fltr_save_file() to determine the file format and compression type you specified in the lFormatType argument (as IG_SAVE_ constant, see accucnst.h file).

Saving Images to Memory

The function IG_fltr_save_mem() allows you to save a HIGEAR image to memory. The result is a file image in memory that is identical to the file that would have resulted if you had used any other save function (such as IG_fltr_save_file()). However, instead of using a filename to call IG_save_mem(), you specify the address and size of the memory area to which to save. The allocation of memory is discussed further below.

If there already is a valid image file at the address you specify in an IG_fltr_save_mem() call, the effect is the same as when using IG_fltr_save_file() to save to an existing file. Specifically, it allows appending or inserting pages into an existing file stored in memory.

Before you call IG_fltr_save_mem(), you need to allocate a memory buffer, and you must supply the size of the allocated buffer to the function. You can determine the appropriate buffer size by making a call to IG_fltr_save_mem_size_calc(). The size returned by this function will include the size of the bitmap data, which can be a portion of the image (the image rectangle) or the whole image, plus any other structures, such as the header or palette. If you are going to add a page to an existing image in the memory buffer, pass the address of the buffer to IG_fltr_save_mem_size_calc(). The function will calculate and return the size necessary for storing the image after the addition of the page.

You can use these steps to save a multi-page file in a memory buffer:

1. Call IG_fltr_save_mem_size_calc(), specifying the HIGEAR for the first page, and passing NULL to lpImage parameter. This will return the size of the first page, saved to the buffer.
2. Allocate a memory buffer using the calculated size.
3. Save first page to the memory buffer, using IG_fltr_save_mem().
4. Call IG_fltr_save_mem_size_calc(), specifying the HIGEAR for the second page, and passing the pointer to the memory buffer you've allocated, to lpImage parameter. This will return the size of the first and second pages saved to the buffer.
5. Reallocate memory buffer using the new size.
6. Save second page.

7. Continue for the rest of pages.

This process can be optimized. For example, you can allocate (size of first saved page) * (number of pages) bytes in the first place to reduce the number of reallocations.

IG_fltr_save_mem() will return the actual size that the file required when it was saved to memory.

See Also

Saving Images to a Disk File

Saving to a Disk File Using a File Descriptor Handle

Saving an Image to Memory

1.2.4.3.3 Format Filter Utility Functions

ImageGear provides a set of API functions that simplify the handling of different file formats in the application. See these topics for additional details:

- Getting Information about a File Format Filter
- Inquiring Format Filters for Supported Features

1.2.4.3.3.1 Getting Information about a File Format Filter

To get information about an ImageGear format filter use the IG_fltr_info_get() function. It returns information about the features supported by ImageGear for this file format (as IG_FLTR_ flags), the short and full names of the filter's file format and the names of the default file's extension:

```
nErrCount = [lpfn]IG_fltr_info_get( nFormat, &dwFlags, szShortName,
sizeof(szShortName), szFullName, sizeof(szFullName), szDefExt, sizeof(szDefExt) );
```

Together with this function, you can use the IG_fltr_formatlist_sort() function to sort file formats in alphabetic order based on the short name returned by the IG_fltr_info_get() function through its third lpShortNameparameter.

If you want to determine the list of filters that support the IG_FLTR_ features you can use the function:

```
IG_fltr_formatlist_get(DWORD dwFlags, LPAT_MODE lpFormatList, UINT nFListSize,
LPUINT lpnFListCount)
```

Through the dwFlagsargument, you can specify any combination of features (for instance, IG_FLTR_DETECTSUPPORT | IG_FLTR_PAGEREADSUPPORT | IG_FLTR_MPAGEREADPSUPPORT), and a list of filters that support those features will be returned through the second parameter (lpFormatList). See the description of this function in the Core Component API Function Reference for detailed information about the parameters. If you want to determine the value of the nFListSizeargument, first set lpFormatListto NULL. You can also use this function together with IG_fltr_formatlist_sort() to get the list of filters that support necessary features and sort them by filter name:

```
UINT                nFilterSize;
UINT                nFListSize;
LPAT_MODE           FList = NULL;
LPCHAR              lpFilter = NULL;
LPSTR              str;
IG_fltr_formatlist_get( IG_FLTR_PAGEREADSUPPORT, NULL, 0, &nFListSize );
FList = malloc( nFListSize*sizeof(AT_MODE) );
if( FList!=NULL )
{
IG_fltr_formatlist_get( IG_FLTR_PAGEREADSUPPORT, FList, nFListSize, NULL );
IG_fltr_formatlist_sort( FList, nFListSize );
...
}
```

You can get information about any page of a multi-page file without loading it into memory if you use the function IG_fltr_pageinfo_get(). Return information consists of the name of the file format (as IG_FORMAT_ constant), the type of image compression (as IG_COMPRESSION_ constant), and such info as bpp, width and height of the page-image.

Another two IG_fltr_...() functions allow you to work with pages in a multi-page file without loading it in memory:

```
IG_fltr_pageswap_file (const LPSTR   lpszFileName, AT_MODE  nFormatType,  UINT
Page1, UINT  Page2)
IG_fltr_pagedelete_file (const LPSTR   lpszFileName, AT_MODE  nFormatType,  UINT
nStartPage, UINT  nRange)
```

Both functions accept the filename of the multi-page image, format filter ID, and two integer parameters as page numbers (assuming numeration from 1). The first function swaps pages in a multi-page image specified by page numbers, but the second function deletes pages from it.

Before using these functions you have to determine whether or not ImageGear supports the corresponding features for the necessary file format. This can be done using the function IG_fltr_info_get() and inspecting flags returned through the second parameter for the specified file formats. If this value contains the flag IG_FLTR_PAGESWAPSUPPORT, then the format filter does support the IG_fltr_pageswap_file() operation. If it also has the flag IG_FLTR_PAGEDELETESUPPORT, then it supports the IG_fltr_pagedelete_file() operation.

SWAP and DELETE operations may or may not physically reorder multi-page files. For example, the page delete operation may not really reduce the size of the file - just update the file format structures and remove the specified pages from it.

1.2.4.3.3.2 Inquiring Format Filters for Supported Features

Each format filter has its own list of supported features determined by IG_FLTR_ flags. The table below describes the meaning of each of these flags:

IG_FLTR_DETECTSUPPORT	Supports auto-detection.
IG_FLTR_PAGEREADSUPPORT	Supports reading a single page file.
IG_FLTR_MPAGEREADPSUPPORT	Supports reading a multi-page file.
IG_FLTR_PAGEINSERTSUPPORT	Supports writing a single-page file.
IG_FLTR_MPAGEWRITEPSUPPORT	Supports writing multi-page file.
IG_FLTR_PAGEDELETESUPPORT	Supports deleting a page from a multi-page file.
IG_FLTR_PAGESWAPSUPPORT	Supports page-swapping in multi-page files.
IG_FLTR_MPDATASUPPORT	Supports faster multi-page access by storing private format data (used only with IG_mpi_... and IG_mpf_... API).

1.2.4.3.4 Working with Multi-Page Documents

Along with a single-page image handle (HIGEAR), ImageGear provides support for multi-page images. The HMIGEAR handle represents an array of single-page images. You can use ImageGear to:

- Create and delete an internal representation of a multi-page image (HMIGEAR handle)
- Open and associate a multi-page image file with an external file
- Access and manipulate pages within the multi-page image
- Manipulate pages in the external image file, such as loading, saving, swapping, and deleting pages
- Retrieve information about multi-page images and about associated external files

The list of functions are divided into two categories:

- Those that work with external associated multi-page files. These functions appear as IG_mpf_... .
- Those that deal with multi-page image arrays. These functions begin with IG_mpi_... .

For example, the function IG_mpf_page_swap() swaps pages in the external file.

This section provides information about the following:

- Creating and Deleting a Multi-page Image
- Opening and Closing an External Image File
- Loading and Saving Pages
- Using Other Functions that Work with Pages
- Using the Multi-Page Image Callback Function

1.2.4.3.4.1 Creating and Deleting a Multi-Page Image Object

An internal representation of a multi-page image is an object of the data type HMIGEAR. This data type encapsulates an array of pages, whose objects are of type HIGEAR. All IG_mpi_*** functions are introduced to access and manage this array.

The first step when working with a multi-page image is to allocate and initialize an object of type HMIGEAR. To do this, call the function IG_mpi_create(). This function creates a new multi-page image and sets the page array size to the given number of pages. Pages are numbered from 0 to nPageCount-1, where nPageCount is the current size of the page array, and the default page value is NULL.

A multi-page image is an array of pages. It is not necessary that all elements of this array contain valid HIGEAR objects. Some pages may have a NULL value, and the function IG_mpi_page_is_valid() can be used to quickly identify whether a page with a given number contains a valid HIGEAR image. The function IG_mpi_page_get() retrieves the value, and the IG_mpi_page_set() function replaces a page value with a given index in the page array of a multi-page image.

The function IG_mpi_page_count_get() returns the current size of a page array, and the function IG_mpi_page_count_set() changes the size of the page array.

When a multi-page image is no longer needed, you should delete it using the IG_mpi_page_delete() function. This function deletes all valid pages in the array and frees all memory allocated in the HMIGEAR handle.

See Also:

Getting Information about a File Format Filter

1.2.4.3.4.2 Opening and Closing an External Image File

After a multi-page image is created, it can be associated with an external image. This allows you to set a relationship between a page array of a given multi-page image and pages of a multi-page file. After making that association, but before closing the file, ImageGear stores format-related information in memory. This allows you to perform page manipulation operations, such as page loads, saves, swaps, or deletes quickly, without scanning all of the format structure of the associated file image. This may be useful if the number of pages in the file is large. It also improves the performance of these operations.

Use the function IG_mpi_file_open() to open or create an external multi-page file and associate it with the multi-page image given by the HMIGEAR object. Use the fourth parameter of this function to specify two open modes. If IG_MP_OPENMODE_READONLY is specified, then the file opened has read-only access. Only these operations are allowed. They need not change the file (for example, page loading).

If IG_MP_OPENMODE_READWRITE is used, then the file is opened with read-write access, and all supported operations are allowed. Not all format filters support such operations, such as page insert, delete, and swap. The function IG_fltr_info_get() can be used to get information about all implemented features of some particular format.

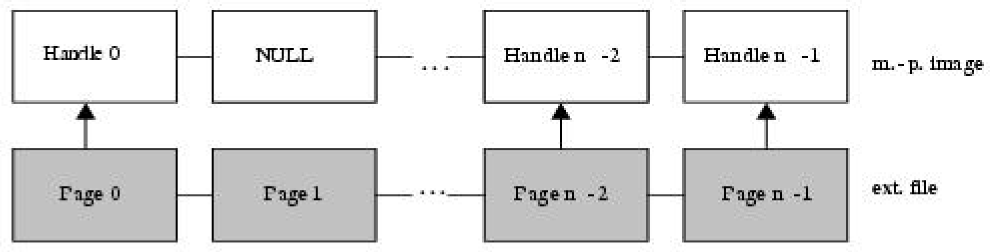
The third parameter of this function provides the format identifier (one from defined IG_FORMAT_... constants in accucnst.h) and is used only when the file image opens in IG_MP_OPENMODE_READWRITE mode. The file does not exist, and it is ignored in other cases. This parameter is used to specify the format of the image to be created.

Example:

```
#include "accucnst.h"
 ...
HMIGEAR hMIGear;         /* HMIGEAR handle returned   */
AT_ERRCOUNT nErrCount;      /* number of errors reported */
 ...
nErrCount = IG_mpi_create(&hMIGear, 0);
if (!nErrCount)
{
nErrCount = IG_mpi_file_open("picture.tif", hMIGear, IG_FORMAT_UNKNOWN,
IG_MP_OPENMODE_READONLY);
         ...
    nErrCount = IG_mpi_close( hMIGear );
}else
         /* error handling */
 ...
```

If this function is opened for read-only access, then the page array of multi-page images is set to the number of pages that is equal to the number of pages in the external file. If the file is opened for read-write access, then the multi-page image is not changed.

Use the function IG_mpi_close() to disassociate a multi-page image from an external file, close the file, and free all correspondent resources.



1.2.4.3.4.3 Loading and Saving Pages

After the multi-page image is associated with an external file, it is possible to perform operations such as page loads, saves, swaps, and deletes. If open mode is read-only, then only the page load is allowed. All others will trigger an error.

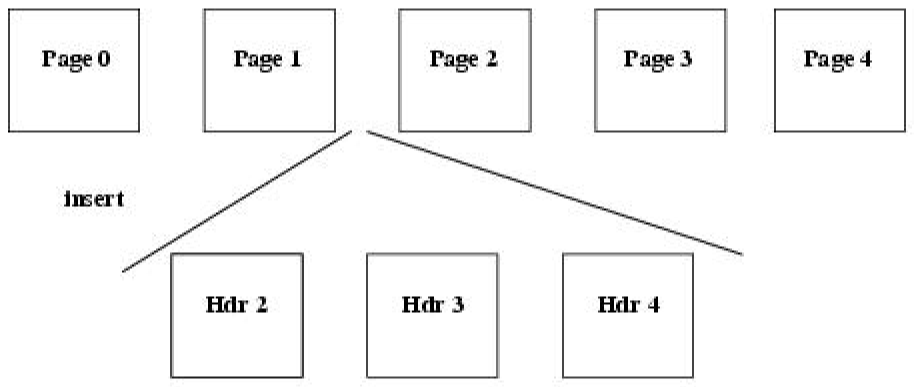
Use the function IG_mpf_page_load() to load pages from an external file into a multi-page image. The second argument of the function is a zero-based index of the first page for loading. The third argument specifies the number of pages to load starting from this index. The order and location of the loaded pages in the page array is the same as in the external file. Therefore, if the file pages start from nFirstIndex, then it loads into the page array starting with nFirstIndex. If necessary, the page arrays are expanded to fit all of the requested number of pages. If, while loading, some elements of the page array contains a valid HIGEAR image, then it is not deleted. It is then assigned a new value of the image loaded from the file.

Use the function IG_mpf_page_save() to save pages from a multi-page image into an external file. The second and third arguments are the same and have the same meaning as the second and third arguments for the load function. The fourth argument specifies the compression method (for example, IG_COMPRESSION_JPEG or IG_COMPRESSION_LZW for a TIFF image) and applies to all pages from the given range. The last parameter of this function specifies how to save the pages into a file. There are two modes are possible:

- IG_MPF_SAVE_INSERT
- IG_MPF_SAVE_REPLACE

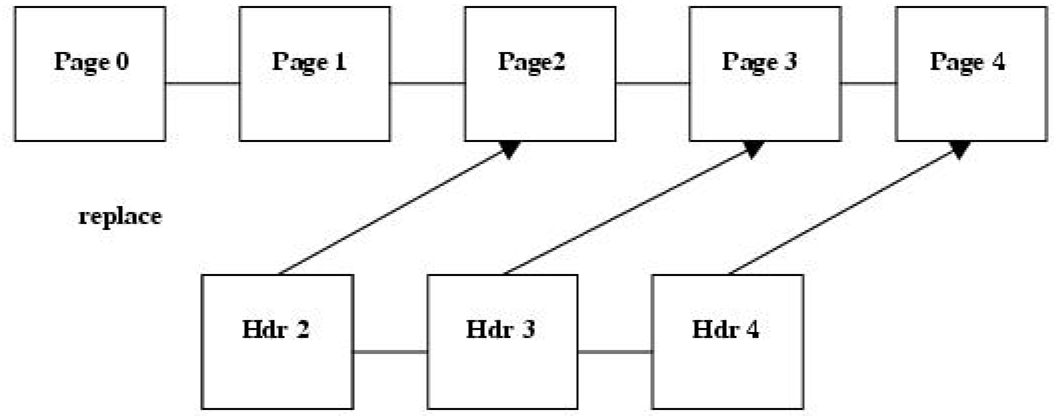
The first mode inserts pages into the file at the location specified by the second parameter, which is demonstrated in the following picture:

IG_MPF_SAVE_INSERT:



The second mode replaces pages starting from the index specified by the second parameter:

IG_MPF_SAVE_REPLACE:



Example:

To load pages 2 through 5 from the file "image1.tif" and insert them into file "image2.tif," the program is:

```
/*...*/
HMIGEAR         hMIGear;
AT_ERRCOUNT        nErrCount;
UINT        nStartPage;
UINT        nCount;
nErrCount = IG_mpi_create(&hMIGear, 0);
if (!nErrCount)
{
nErrCount = IG_mpi_file_open( "picture1.tif", hMIGear, IG_FORMAT_UNKNOWN,
IG_MP_OPEN_READ );
        nStartPage = 2;
        nCount = 4;
        nErrCount = IG_mpf_page_load( hMIGear, nStartPage, nCount );
}
if (!nErrCount)
{
        nErrCount = IG_mpi_close( hMIGear );
        if (!nErrCount)
            nErrCount = IG_mpi_file_open( "picture2.tif", hMIGear, IG_FORMAT_UNKNOWN,
IG_MP_OPEN_READWRITE );
    if (!nErrCount)
    nErrCount = IG_mpf_page_save( hMIGear, nStartPage, nCount,
IG_COMPRESSION_JPEG, IG_MPF_SAVE_INSERT );
}
if (nErrCount)
{
        /* error handling */
#include "accucnst.h"
}
```

If a page from the pages to save is not a valid HIGEAR image, then this page is ignored during the save operation.

IG_MPF_SAVE_REPLACE mode is acceptable for filters that support the page deletion operation. This information can be obtained using the function IG_fltr_info_get(), and it should return the IG_FLTR_PAGEDELETESUPPORT flag, which is set in the dwInfoFlags parameters.

1.2.4.3.4.4 Using Other Functions that Work with Pages

When an external file is opened in read-write mode, an operation such as page delete and page swap is possible. Not all filters support it. It is necessary to check with the function IG_fltr_info_get() for flags IG_FLTR_PAGEDELETESUPPORT and IG_FLTR_PAGESWAPSUPPORT in the lpdwInfoFlags parameter.

IG_mpf_page_swap() is used to reorder pages in the external file. The second and third arguments of this function are zero-based indexes of the pages to be swapped.

The IG_mpf_page_delete() function deletes the specified pages from the external file. The index of the first deleted page is passed through the second argument, and number of pages to be deleted is passed through the third argument.

Example:

This example shows how to reorder pages in a multi-page file.

```
AT_ERRCODE mpfPageReorder(
        HMIGEAR hMIGear
)
{
        UINT nPageCount;
        AT_MODE nFormatID;
        AT_ERRCOUNT nErrCnt;
        DWORD dwInfoFlags;
        UINT i;
IG_mpf_info_get( hMIGear, &nFormatID );
        if( nFormatID==IG_FORMAT_UNKNOWN )
        return  -1; /* file is not associated */
        IG_fltr_info_get( nFormatID, &dwInfoFlags, NULL, 0, NULL, 0, NULL, 0 );
        if( (dwInfoFlags&IG_FLTR_PAGESWAPSUPPORT)==0 )
        return  -1;/* format filter does not support the page swap operation */
        nErrCnt = IG_mpf_page_count_get( hMIGear, &nPageCount );
        for( i = 0; (i < nPageCount/2) && (nErrCnt==0); i++ )
nErrCnt = IG_mpf_page_swap( hMIGear, i, nPageCount - i - 1 );
        return -nErrCnt;
}
```

Example:

This example demonstrates how to perform this operation on a multi-page image located in memory:

```
AT_ERRCODE mpiPageReorder(
HMIGEAR hMIGear
{
HIGEAR hPage1, hPage2;
UINT i, nPageCount;
AT_ERRCOUNT  nErrCnt = 0;
nErrCnt = IG_mpi_page_count_get( hMIGear, &nPageCount );
for( i = 0; (i<nPageCount/2) && (nErrCnt==0); i++ )
        {
        nErrCnt = IG_mpi_page_get( hMIGear, i, &hPage1 );
        nErrCnt += IG_mpi_page_get( hMIGear, nPageCount - i - 1, &hPage2 );
        nErrCnt += IG_mpi_page_set( hMIGear, i, hPage2 );
        nErrCnt += IG_mpi_page_set( hMIGear, nPageCount - i - 1, hPage1 );
}
return -nErrCnt;}
```

See Also:

IG_fltr_pageswap_file()

IG_fltr_pagedelete_file()

Getting Information about a File Format Filter

1.2.4.3.4.5 Using the Multi-page Image Callback Function

Multi-page image operations, implemented by the above functions, support a notification mechanism that allows you to track information about when and how the multi-page image or associated file was changed. This can be done using the functions IG_mpi_CB_set(), IG_mpi_CB_get(), IG_mpi_CB_reset(), and IG_mpi_CB_reset_all().

Use the function IG_mpi_CB_set(hMIGear, lpPrivate, lpfnUpdate, lpdwCBID) to associate new callback data with the given hMIGear handle. The second argument, lpPrivate, is any LPVOID pointer that the callback function lpfnUpdate receives through the second parameter. lpfnUpdate is a pointer to the function that implements the following interface:

```
typedef  VOID (LPACCUAPI LPFNIG_MPCB_UPDATE)(
       DWORD dwCBID,
       LPVOID lpPrivate,
       AT_MODE nMode,
       UINT nPage,
       UINT nCount
);
```

The last argument of IG_mpi_CB_set(), lpdwCBID, is a pointer to the application that receives an unique DWORD identifier for the associated callback data. This ID is used to delete callback data using the function IG_mpi_CB_reset() and retrieves callback data using the function IG_mpi_CB_get(). After the IG_mpi_CB_set() function is executed, ImageGear calls the lpfnUpdate function every time a multi-page image is changed. This allows the application to react accordingly, and updates the related objects such as GUI windows. As soon as this callback function is called from the context of the thread that performed the operation, its execution is blocked until the callback function is complete. From one side, this can be used for synchronization; but from another side, it should be used carefully so that it does not affect performance.

By calling the callback function, ImageGear passes the type of changes through the nMode argument. The sense of nPage and nCount arguments depend upon nMode. The following table list all possible cases:

nMode	nPage	nCount	Description
IG_MPCBMODE_MPI_DELETE	Not used	Not used	Notifies the application that a multi-page image is going to be deleted.
IG_MPCBMODE_MPI_ASSOCIATED	Not used	Not used	Notifies the application that a multi-page image is just associated with external file.
IG_MPCBMODE_MPI_CLOSE	No used	Not used	Notifies the application that a multi-page image is going to close the associated external file.
IG_MPCBMODE_MPI_CB_SET	Not used	Not used	Notifies the application that this callback data is just set. This notification receives only the callback function that just has been set.
IG_MPCBMODE_MPI_CB_RESET	Not used	Not used	Notifies the application that this callback data is to be reset.
IG_MPCBMODE_MPI_PAGEINSERTED	Index of where new pages start	Number of new pages inserted	Notifies the application that new pages are inserted into the multi-page image.
IG_MPCBMODE_MPI_PAGEUPDATED	Index of the first updated page	Number of updated pages starting from nPage	The application updated pages in the multi-page image.
IG_MPCBMODE_MPI_PAGEDELETED	First deleted page index	Number of deleted pages	The application deleted pages in the multi-page image.
IG_MPCBMODE_MPF_PAGEINSERTED	Index of where new pages start	Number of new pages inserted	The application inserted new pages into the external file image.

IG_MPCBMODE_MPF_PAGEUPDATED	Index of the first updated page	Number of updated pages starting from nPage	The application updated pages in the associated external multi-page image file.
IG_MPCBMODE_MPF_PAGEDELETED	Index of the first deleted page	Number of deleted pages	The application deleted pages in the associated external multi-page image file.

When the application does not need to receive any more information from the callback data, it should call the function IG_mpi_CB_reset(hMIGear, dwCBID), where dwCBID is a unique identifier of the association returned by IG_mpi_CB_set().

The function IG_mpi_CB_reset_all() removes all callback data associated with all previously allocated identifiers.

See Also:

Using Filter Callback Functions to Process Non-Image Data

Working with ImageGear Callback Functions

1.2.4.3.5 Format Filter Control Parameters

Almost every format filter in ImageGear has some attributes on which it depends while processing operations such as READ, WRITE, etc. Those attributes may be attributes declared by the format filter specification or may be specific to its implementation by ImageGear. ImageGear has a general public interface implemented and named as format filter control parameters. Every such control parameter is identified by the format filter and string name and has an associated type of acceptable value, the value itself, and the default value. Each control parameter is filter specific. The ImageGear Supported File Formats Reference describes the filters, and also describes each control parameter for each format filter.

The ImageGear Filters API has three functions that allow you to get/set info about every supported filter control parameter:

- IG_fltr_ctrl_list(DWORD dwFormatID, LPUINT lpnCount, LPDWORD lpArray, DWORD dwArraySizeInBytes)
  This function allows the application to get the list of names of all control parameters supported by the format filter identified by dwFormatID. The Application is responsible for allocating the buffer lpArray, but ImageGear sets the elements of this array as pointers to strings with control parameter names. You can use the control parameter names as input values for the second argument of the next two functions.
- IG_fltr_ctrl_get(DWORD dwFormatID, const LPCHAR lpcsCtrlName, AT_BOOL bGetDefault, LPAT_MODE lpnValueType, LPDWORD lpdwValueSize, LPVOID lpBuffer, DWORD dwBufferSize)
  This function is used to get the value of a given control parameter of a given format filter, and it may return either its current value or its default value - that is controlled by the bGetDefault argument. A TRUE value returns the default value, but FALSE returns the current value. The value itself is copied into a buffer that the application provides through the dwBufferSize argument. You can use the control parameter names from IG_fltr_ctrl_list as input values for the lpcsCtrlName argument.
- IG_fltr_ctrl_set(DWORD dwFormatID, const LPCHAR lpcsCtrlName, LPVOID lpValue, DWORD dwValueSize)
  This function allows you to set a new value for the control parameter. You can use the control parameter names from IG_fltr_ctrl_list as input values for the lpcsCtrlName argument. The last two arguments of this function specify the data to be set, and ImageGear always treats this data as a type that can be gotten by the _get() function. If the actual size of the new value is less than 4 bytes, then lpValue is treated as the value itself, otherwise it is treated as a pointer to the value.

This example demonstrates how to get the names of all supported control parameters for the TIFF format filter:

```
/* getting the total number of parameters */
nErrCount = IG_fltr_ctrl_list(IG_FORMAT_TIF, &nCount, NULL, 0);
if(!nErrCount && nCount > 0)
{
/* allocate required buffer to keep all names */
lpOptList = malloc(nCount * sizeof(DWORD));
        if(lpArray)
        {
        nErrCount = IG_fltr_ctrl_list(IG_FORMAT_TIF, NULL, lpOptList, nCount *
sizeof(DWORD));
```

The filter control parameters you work with using IG_fltr_ctrl_...() functions are strings. Refer to the "Filter Control Parameters" Tables for each file format in the ImageGear Supported File Formats Reference section.

This example demonstrates how to get and set the value of the TIFF control parameter named "BIG_ENDIAN":

```
char DocumentName[_MAX_PATH];
AT_BOOL bDefBigEndian, bOldBigEndian;
...
/* get current value of BIG_ENDIAN control parameter */
IG_fltr_ctrl_get(IG_FORMAT_TIF, "BIG_ENDIAN", FALSE, NULL, NULL,
(LPVOID)&bOldBigEndian, sizeof(hOldBigEndian));
/* get default value of BIG_ENDIAN control parameter */
IG_fltr_ctrl_get(IG_FORMAT_TIF, "BIG_ENDIAN", TRUE, NULL, NULL,
(LPVOID)&bDefBigEndian, sizeof(hDefBigEndian));
/* get current value of DOCUMENT_NAME control parameter */
IG_fltr_ctrl_get(IG_FORMAT_TIF, "DOCUMENT_NAME", FALSE, NULL, NULL, DocumentName,
sizeof(DocumentName));
/* set new value to BIG_ENDIAN control parameter */
```

```
IG_fltr_ctrl_set(IG_FORMAT_TIF, "BIG_ENDIAN", (LPVOID)TRUE, sizeof(AT_BOOL));
/* set new value to DOCUMENT_NAME control parameter */
strcpy( DocumentName, "This is a test string for DocumentName" );
IG_fltr_ctrl_set(IG_FORMAT_TIF, "DOCUMENT_NAME", (LPVOID)DocumentName,
sizeof(DocumentName));
```

For the TXT Filter, to set the LINES_PER_PAGE and CHAR_PER_LINE control parameters, set the POINT_SIZE control parameter to zero; setting the PAGE_WIDTH, PAGE_HEIGHT, and POINT_SIZE parameters provides a sufficient page description, and LINES_PER_PAGE and CHAR_PER_LINE options are ignored.

1.2.4.3.6 Non-Image Data Processing

Some format filters, such as EXIF-JPEG, EXIF-TIFF, TIFF, JPEG, PNG and some others contain non-image data, generally referred to as metadata. ImageGear provides a mechanism for reading the metadata during image loading and modifying it during image saving. Non-image data itself can be of any possible complex type, depending on the nature of the file format. ImageGear processes this complex data through a single interface and allows uniform processing that does not depend on the actual data format, and starts from information fields of such simple formats as BMP and PCX, up to the complex metadata support in the EXIF filter and the IPTC non-image data format in such filters as TIFF and JPEG.

While EXIF-JPEG and EXIF-TIFF are separate image file formats using JPEG or TIFF image data compressions, IPTC is a format used only for non-image data storage in such imaging format filters as JPEG and TIFF. For more detailed information about these formats, see the EXIF-TIFF Non-Image Data Structure, EXIF-JPEG Non-image Data Structure, and the IPTC Non-Image Data Structure sections in Non-Image Data Storage as well as the EXIF-JPEG and EXIF-TIFF sections in the File Format Reference.

ImageGear is responsible for translating this format-dependent data into a standard uniform format. There are at least two operations that include such data processing:

- Image loading - during image loading, some additional data needs to be loaded and uncompressed into the set of values of standard types for further processing.
- Image saving - during image saving, there should be a way to change existing defined values and add new values.

This section provides the following information:

- Non-Image Data Format
- Using Filter Callback Functions to Process Non-Image Data
- Updating Non-Image Data without Loading and Saving the Image
- Working with XMP Metadata

1.2.4.3.6.1 Non-Image Data Format

The key thing of non-image data processing in ImageGear is a uniform data format that is used to convert to and from the format filter. As soon as the format filter decodes the data fields one after another during the loading operation, and encodes it in the reverse direction during the saving operation, all data consists of the set of items where each item is a minimal atom of information. The order of items is fixed, and the format filter processes item after item in the given order. The same order is used when data is passed through the stream.

The low-level format of the data consists of the list of items where each item represents a minimal unit of information. Each item also should have some unique name that allows you to connect it with the physical value inside of the file format. The definition of the data item can be described by the following fields:

```
typedef struct tagAT_DATALIST_ITEM{
    AT_MODE         FormatID
    LPCHAR          Name;
    DWORD          Id;
    AT_MODE          Type;
    LPVOID          Value;
    AT_MODE          ValueType;
    DWORD          Length;
    AT_MODE          ValueAccessMode;
    }AT_DATALIST_ITEM;
```

Please see the descriptions of these fields below:

FormatID	The ID of the filter that reads or writes a file (IG_FORMAT_... constant value).
Name	The name of the item. Can be any string value.
Id	Numerical ID of item. Can be any value of DWORD size.
Type	Specifies the type of item and reflects the status of the given record. Possible values are: • IG_METAD_VALUE_ITEM - this value specifies that the current item is a value of the simplest type, and the field Value contains the actual value of the item, and ValueType contains the identifier of the type of this item. ReadOnly can be either TRUE (read-only) or FALSE (read/write). Name and/or Id contains textual and numerical identification of the item. • IG_METAD_LEVEL_START - this value specifies that the current item opens the sublevel of items and all the next items up to the corresponding item with the LEVEL_END value belonging to this sublevel. • IG_METAD_LEVEL_END - this value closes the current sublevel and tells that next item belongs to a higher level.
Value	Contains the value of the item when Type = IG_METAD_VALUE_ITEM. Note that possible values of this field are fixed and define the exact list of allowed data types. It also depends on the ImageGear platform and FLTR.METADATA_FORMAT global control parameter. This global parameter has two allowed values: "text" and "binary". See the section Metadata Structure "ValueType" and "Value" for possible values.
ValueType	Contains the type identifier of the item when Type = IG_METAD_VALUE_ITEM. Possible values of this field are fixed and define the exact list of allowed data types. See the section Metadata Structure "ValueType" and "Value" for possible values.
Length	Identifies the number of values to be written. • For AM_TID_TXT_STRING it should indicate the number of characters in the string, excluding last null character (basically the length of the string). • For AM_TID_RAW_DATA it should indicate the number of bytes that the raw data occupies. • For the rest of the types it should indicate the number of values of the type, which textual representation is encoded into "Value".
ValueAccessMode	Identifies whether data can be changed or not. "Read only" value means that its value is information only and cannot be changed after setting the initial value. It also means that its value will be ignored during a WRITE operation.

So, this data structure allows you to "linearize" hierarchical and complex data into an array of simplest data types.

You can transfer different non-image data using the general data structure described in this section. Please see Non-Image Data Storage.

1.2.4.3.6.2 Using Filter Callback Functions to Process Non-Image Data

There are two working scenarios of how ImageGear processes non-image data.

The first one is the LOAD operation:

1. Application registers special callback function of type LPAFT_IG_METAD_ITEM_GET_CB.
2. The application calls some of the filter loading functions (like IG_fltr_load_file()), and during the LOAD operation, the format filter calls the registered callback function to pass data for each item decoded from the image.

The reverse WRITE operation is more complex:

1. Application registers callback functions of types LPAFT_IG_METAD_ITEM_SET_CB and LPAFT_IG_METAD_ITEM_ADD_CB.
2. Application calls some filter writing functions (like IG_fltr_save_file()).
3. While performing WRITE operation ImageGear uses callback functions to modify existing items or add additional items to required dataset.

> ImageGear provides special LPAFT_ callback functions for the non-image data processing described in this section. It also preserves the "old" callback functionality (LPFNIG_ callback functions) required for image processing control and perfection. Please see Working with ImageGear Callback Functions for detailed information about the structure of the ImageGear callback functionality.

You can see from the declaration below that LPAFT_IG_METAD_ITEM_GET_CB accepts parameters that provide all necessary information about one data item. All parameters except the first one are fields of the data structure AT_DATALIST_ITEM described in Non-Image Data Format:

```
LPAFT_IG_METAD_ITEM_GET_CB(LPVOID lpPrivate, LPCHAR ItemName,  DWORD ItemID, AT_MODE
ItemType, LPVOID ItemValue, AT_MODE ValueType, DWORD ValueLength, AT_BOOL
ReadOnlyValue )
```

By implementing and providing a callback function of this type, the application can receive every decoded item and process it as needed.

Some items from the dataset are informational only and cannot be changed during the WRITE operation. So, if the ReadOnlyValue field is set to TRUE, then the item will not be changed during the WRITE operation. Actually, during this operation, the format filter prepares all necessary items and puts required values to them to make sure that the file format itself is not violated. For example, if an item requires a particular number of strips in the image, then if the value of this item is changed, the image cannot be loaded.

The format filter prepares a minimal set of items and default values for them, and before writing its values to the output stream it calls the callback function of type LPAFT_IG_METAD_ITEM_SET_CB so that the application can change its values. In addition, the format filter may call the function of type LPAFT_IG_METAD_ITEM_ADD_CB to get additional items to append the dataset. The application should provide its implementation in such a way that it returns TRUE until it is necessary to insert more items. If this function returns FALSE, all custom items have been added, and the filter can proceed.

> It may happen that the application provides an item with the name or ID of a different type than the format filter is expecting. For example, the format filter may expect the item named SOFTWARE with text as a String, but the application provides its value as an Integer. In this case the filter may ignore this item and trigger a warning that this type of item is not expected. Also, the application may provide an item that the format filter is not able to handle because it does not fit the format defined by the corresponding image file format. In this case it may simply ignore the item and give a warning.

The exact specification of the callback function types can be found in Core Component API Function Reference, but the following is a quick reference for better understanding:

```
LPAFT_IG_METAD_ITEM_SET_CB)(LPVOID lpPrivate, LPCHAR ItemName, DWORD ItemID, AT_MODE
ItemType, LPVOID ItemValue, AT_MODE ValueType, DWORD ValueLength, AT_BOOL
ReadOnlyValue, LPVOID *NewItemValue, LPAT_MODE *NewValueType, LPDWORD
*NewValueLength )
```

*NewItemValue, *NewValueType, *NewValueLength are arguments with a new value for a given item. If

ReadOnlyValue is TRUE, the value of this item is unchangeable.

You can add a new non-image item during the filter WRITE operation using the callback function prototype:

```
LPAFT_IG_METAD_ITEM_ADD_CB(LPVOID lpPrivate, LPCHAR ItemName,  DWORD ItemID, AT_MODE
ItemType, LPVOID ItemValue, AT_MODE ValueType, DWORD ValueLength, AT_BOOL
ReadOnlyValue )
```

All arguments in this function are parameters for the new item and its value.

To exchange the non-image tag information provided by these three callback functions between the application and the internal ImageGear structure levels, you should use two functions:

```
IG_fltr_metad_callback_get(LPVOID *lpPrivate, LPAFT_IG_METAD_ITEM_SET_CB
*lplpfnSetCB, LPAFT_IG_METAD_ITEM_ADD_CB  *lplpfnAddCB,
LPAFT_IG_METAD_ITEM_GET_CB *lplpfnGetCB )

IG_fltr_metad_callback_set(LPVOID *lpPrivate, LPAFT_IG_METAD_ITEM_SET_CB
*lplpfnSetCB, LPAFT_IG_METAD_ITEM_ADD_CB  *lplpfnAddCB,
LPAFT_IG_METAD_ITEM_GET_CB *lplpfnGetCB )
```

The first function allows you to provide the current callback non-image tag data from the internal ImageGear structure to an application level during load/save processes. If some callback information is not necessary, you can set the respective argument of this function to NULL. For instance, if you do not need information about newly added non-image items, set lplpfnAddCB = NULL.

The second _set() function provides the new non-image callback data from the application level to the internal ImageGear level during the load/save processes. Again, if some callback information is not necessary, you can set the respective argument of this function to NULL.

An example of how to use these callback functions is too complex to include in this manual. The GUI implementation that is provided in the source form demonstrates all aspects of working with these callback functions.

1.2.4.3.6.3 Updating Non-Image Data without Loading and Saving the Image

Callback functions can be used to get and set non-image data during normal load and save operations. It is also possible to use callback functions with the IG_fltr_metad_update_file() function to operate on only the non-image data in a file. This function creates a new file with an exact copy of the source file's pixel data and with new non-image data. Pixel data is not decoded, but is copied directly from the source to the destination file. This function is currently supported with the following file formats only:

- TIFF (except TIFF-JPEG)
- JPEG

IG_fltr_metad_update_file() obtains new non-image data from the following callback functions:

- LPAFT_IG_METAD_ITEM_SET_CB
- LPAFT_IG_METAD_ITEM_ADD_CB

IG_fltr_metad_update_file() function can be used as follows:

- Load necessary page to get metadata.
- Change metadata (add / delete / change metadata tags or metadata values).
- Call IG_fltr_metad_update_file() function. nPageNumber and lFormatType parameter values should correspond to the loaded page and source file format.

The destination file will be a copy of the source file with the new non-image data for the specified page.

The application may then delete the source file and rename the destination file with the name of the source file.

Usage of metadata update function is demonstrated in the Filter sample. Use the following steps to test this feature:

1. Open a TIF image using File/Open menu item.
2. Add or modify tag(s) using Image/Metadata/Data Structure dialog.
3. Create a file with updated metadata using File/Update metadata menu item. Source file name and page number correspond to the last loaded page in the filter sample.

All changes made in the Metadata dialog will be lost if any function that returns metadata is called before calling IG_fltr_metad_update_file() (i.e., loading a new file, getting file info, or preview).

1.2.4.3.6.4 Working with XMP Metadata

Extensible Metadata Platform (XMP) is an XML-based standard for storage and interchange of metadata, developed by Adobe Systems Inc. The standard defines the rules for storage and processing of the metadata, and provides a number of schemas for storage of information that is typically associated with images and documents, such as Title, Author, Creation date/time, Rating, etc. Applications can add their own schemas to store arbitrary information.

XMP metadata can be attached to files of various formats, such as TIFF, JPEG, PSD and PDF, or stored as a standalone file.

ImageGear provides the following ways for working with XMP metadata:

- Accessing XMP properties via the ImageGear Metadata API. In this mode, ImageGear decodes XMP properties and sends them to the application via metadata callbacks.
- Working with unprocessed XMP metadata. In this mode, ImageGear passes XMP to the application as a byte array, treating it as a single tag of the containing metadata format.

By default, ImageGear parses the XMP stream into its metadata structure, and does not provide the Raw XMP stream. If you don't want ImageGear to parse the XMP stream, and prefer to instead access the unprocessed XMP stream, set global control parameter XMP.Parse to FALSE.

During saving, if XMP.Parse is TRUE, ImageGear expects a tree under the XMP tag, and serializes this tree into the output file. If XMP.Parse is FALSE, ImageGear expects a byte array in the XMP tag, and saves it verbatim to the file.

Since the XMP standard identifies schema properties using string names, rather than numbers, ImageGear also uses names to identify XMP properties. However, it uses numeric identifiers to differentiate between the kinds of entities, such as Description (schema), Array, Property, Qualifier, etc. All properties have ID = ImGearXMPTagIDs.Property, but differ by their names.

See XMP Non-Image Data Structure for a description of XMP metadata structure in ImageGear.

Example:

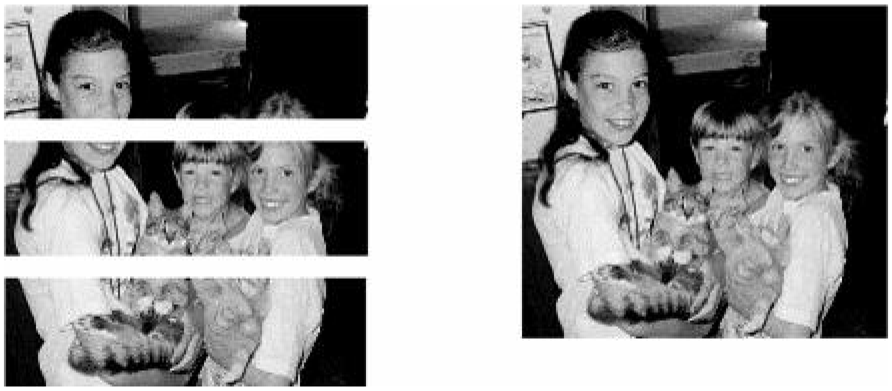
Representation of XMP metadata tree.

- XMP
  - http://ns.adobe.com/xap/1.0/
    - About = ""
    - Namespace
      - Prefix = "xmp"
      - URI = "http://ns.adobe.com/xap/1.0/"
    - Properties
      - xap:Rating
        - Value = 4
      - xap:Identifier
        - Value = "1234.0345.34532.234231"
  - http://purl.org/dc/elements/1.1/
  - Namespace
    - Prefix = "dc"
    - URI = http://purl.org/dc/elements/1.1/
  - Properties
    - dc:subject
      - Bag
    - Item
      - Value = "Test subject 1"
    - Item
      - Value = "Test subject 2"
    - dc:title
      - Alt
    - Item
      - Lang = "x-default"
      - Value = "XMP Support Specification"

1.2.4.3.7 Stripped Images

To expedite loading and displaying large images, ImageGear provides alternate storage schemes that allow a large image to be loaded in segments. Such segments are loaded and pieced together in sequential order.

One of the storage schemes developed was stripped storage. A stripped image contains offsets to numerous groups of pixel rows. Each strip is the full width of the image. The diagrams below show an image stored in strips (shown separated), and the same image pieced together in memory:



In the actual DIB, there are no spaces between strips. The header simply contains the address of the leftmost pixel of the first row of each strip. The spaces between the strips in the image on the left are just for illustrative purposes. On the right, the image has been loaded and pieced together in memory.

The TIFF format, in which the use of stripped storage was once commonplace, rarely uses this scheme today. However, there are many old images still in use that were created with this storage scheme, and ImageGear fully supports these images.

The loading and saving of stripped images using ImageGear is no different than loading or saving a non-stripped image. You can use IG_load_file() or any other ImageGear loading API, without making any special settings. When loading, ImageGear will automatically detect the stripped storage and piece all of the strips together in the proper order.

Once in memory, a stripped image will be like any other image loaded into memory. It will be a DIB with a HIGEAR handle. You can now save it to any ImageGear-supported format. If you would like to save it again using stripped storage, you may do so using ImageGear Format Filter IG_fltr_ctrl_set()/IG_fltr_ctrl_get() functions. Of course, you must choose a file format that supports stripped storage. You can save a TIFF image with stripped storage by setting the number of strips to use. Here is the call you would make:

```
/* get current value of NUMBER_OF_STRIPS control parameter */
IG_fltr_ctrl_get(IG_FORMAT_TIF, "NUMBER_OF_STRIPS", FALSE, NULL, NULL,
(LPVOID)nNumberOfStrips, sizeof(nNumberOfStrips));
/* set new value to NUMBER_OF_STRIPS control parameter */
IG_fltr_ctrl_set(IG_FORMAT_TIF, "NUMBER_OF_STRIPS", (LPVOID)3,
sizeof(nNumberOfStrips));
```

In fact, there are a number of ways that a TIFF may be stored using IG_fltr_ctrl_set() with WRITE_CONFIG parameter. See this control parameter and its range of settings in TIFF Control Parameters Table located in the ImageGear Supported File Formats Reference.

TIFF filter control parameters that pertain to stripped storage include those shown in the following list:

- NUMBER_OF_STRIPS
- BUFFER_SIZE
- WRITE_CONFIG

Please see the section TIFF in the File Format Reference for more detailed information about each parameters.

1.2.4.3.8 Tiled Images

In a tiled image, pixel data is stored in blocks called "tiles", whose height and width are less than that of the height and width of the full image. All tiles are the same size. However, it is rare that the tiles evenly cover the image. Therefore, in most cases, the following statements will be TRUE:

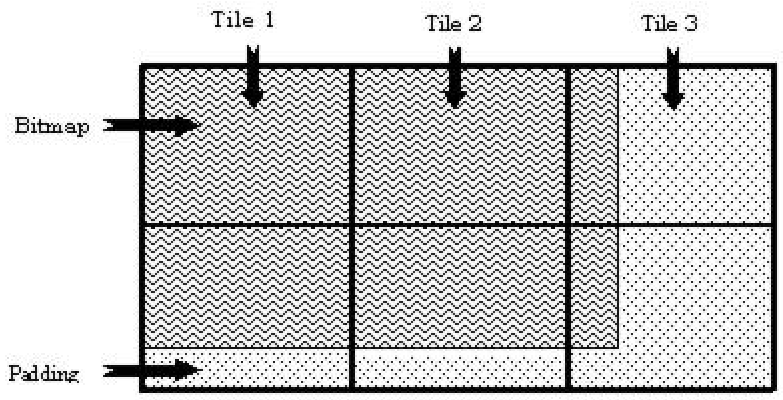
```
(# of tiles per row) * (width of a tile)  ...  the width of the image
(# of tiles per column) * (height of a tile)  ...  the height of the image
```

This section provides information about the following:

- Padding
- Automatic Tile Stitching
- Saving a TIFF File Using Tiles

1.2.4.3.8.1 Padding

Those areas of tiles that are on the border of an image, but are not completely filled with image data, are filled with padding. The image below illustrates a tiled image and shows padding to the right side and bottom of the image. Padding to the right and bottom of the image is the most frequent tiled storage situation that you will encounter:



When ImageGear loads a tiled image (or any image) it reads the header or tag data to find out the width and height of the actual image. This data will not include the padding. As ImageGear loads the file, it discards the padding. The following ImageGear-supported file formats allow the use of tiled storage: TIFF, IBM IOCA, and IBM MO:DCA. Currently, the highest level of control given to you for working with tiled images is for the TIFF format, which will be mentioned frequently throughout this section.

If you were to call IG_load_file() to load a tiled image, it would load the first tile of the first page. If you want to load more than one tile and stitch them together, you should call one of the IG_load_tiles_stitch ...() functions. In addition to loading and stitching any number of tiles, ImageGear also allows you to query the tiles of an image before loading it, tell ImageGear which tiles to "stitch together" when loading, and how to tile the image when you save it to disk.

If you wish to specify which tiles of a TIFF image to load and stitch, your first task will be to find out whether the image actually has tiles or not. The function IG_tile_count_get() can be called to get this information. This function returns the number of tiles per row, and the number of tiles per column. If this function returns zeros for the number of tiles across the image (lpTileCols) and the number of tiles down the length of the image (lpTileRows), you will know that the image is not tiled. There are two other versions of this function: if you have already opened a TIFF file and have a File Descriptor handle for it, use the function IG_tile_count_get_FD(); if the file has already been loaded into memory (using IG_load_mem()), use the function IG_tile_count_get_mem().

The following ImageGear functions can be used to load and stitch TIFF tiles:

```
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch(const LPSTR lpszFileName, UINT nPage,
LPAT_STITCH lpStitch, LPHIGEAR lphIGear);
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch_FD(INT fd, LONG lOffset, UINT nPage,
LPAT_STITCH lpStitch, LPHIGEAR lphIGear);
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch_mem(LPVOID lpImage, DWORD dwImageSize, UINT
nPage, LPAT_STITCH lpStitch, LPHIGEAR lphIGear);
```

Each function takes a page number that specifies which page of a multi-page file to load. Set this to 1 if it is not a multi-page file. Each also takes a structure of type AT_STITCH that will tell ImageGear which tiles to stitch together and load. Here is the definition of AT_STITCH:

```
typedef struct tagAT_STITCH
{
LONG uRefTile; /* Upper left-hand corner tile # */
LONG uTileRows; /* Number of tiles to stitch across*/
LONG uTileCols; /* Number of tiles to stitch down */
}AT_STITCH, FAR *LPAT_STITCH;
```

Set the structure member uRefTile to the number of the tile that you would like to be used as the upper left-most tile in the image that will be loaded and stitched. Set uTileRows to the number of tiles across that you would like your stitched image to use. Set uTileCols to the number of columns of tiles that you would like your stitched image to use. If you wish to stitch all tiles together, simply pass in a NULL for this structure.

Below are some examples of how a tiled image can be stitched together. Figure 1 shows a tiled TIFF image that contains a total of 9 tiles. Note that for simplicity, the tiles are shown to evenly cover the exact height and width of the image. This rarely happens.

The numbering scheme in Figure 1 reflects the way that ImageGear keeps track of the tiles. When you refer to specific tiles, the upper-left-most tile will be 1. The tiles are then numbered sequentially from left to right, top to bottom.

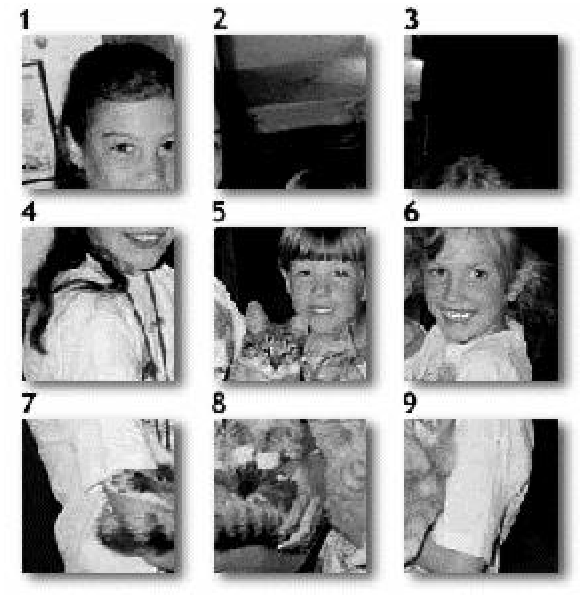


Figure 1: TIFF image with 9 tiles. The numbering scheme shown is the same one you should use when interfacing with tiles using ImageGear.
Figure 2 shows all of the tiles stitched together, which would be the result if you set the AT_STITCH structure to NULL.

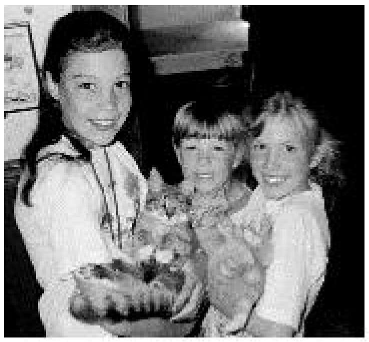


Figure 2: All of the tiles are loaded and stitched together.
The following call achieves the results shown in Figure 2:

```
IG_load_tiles_stitch("KidsKatz.tif", 1, NULL, &hIGear)
```

Figure 3: Tiles 5, 6, 8, and 9 are loaded and stitched together.

To achieve the results shown in Figure 3, make the following calls:

```
AT_STITCH stitchStruct;
HIGEAR hIGear;
stitchStruct.uRefTile = 5;
stitchStruct.uTileRows = 2;
stitchStruct.uTileCols = 2;IG_load_tiles_stitch("KidsKatz.tif", 1, &stitchStruct, &hIGear)
```

You may want to find out the width and height of the tiles in the image so that you can make an informed decision about which tiles to load. First use the function IG_info_get_ex() which will return the width and height of the image. Then divide the width of the image by the number of tiles per row, and the height of the image by the number of tiles per column. To get the height and width of an image that is already opened and for which you have a File Descriptor handle, call IG_info_get_FD_ex(); to get the height and width of an image that has been loaded into memory and for which you have a HIGEAR handle, call IG_info_get_mem_ex().

1.2.4.3.8.2 Automatic Tile Stitching

ImageGear allows you to automatically stitch all image tiles during loading. Set the "STITCH_TILES" filter control parameter to TRUE to enable automatic tile stitching. In this mode:

- Image loading functions load the whole image as if it were not tiled.
- Tile counting functions report that the image is not tiled (has a single tile).
- Header reading functions report full image dimensions rather than dimensions of a single tile.

Set the "STITCH_TILES" filter control parameter to FALSE to disable automatic tile stitching. In this mode:

- Image loading functions load a single tile (by default - first image tile).
- Tile counting functions report the actual number of tiles.
- Header reading functions report the dimensions of a single tile.

Currently, the following ImageGear format filters support automatic tile stitching:

- IBM AFP
- IBM IOCA
- IBM MO:DCA
- JPEG 2000
- JPX
- TIFF

1.2.4.3.8.3 Saving a TIFF File Using Tiles

When you save a TIFF image to disk, you can tell ImageGear to save the bitmap data as tiles, regardless of what storage scheme was originally used. To choose tiled storage and to make modifications to the way ImageGear will implement these storage schemes, use the IG_fltr_ctrl_set() and IG_fltr_ctrl_get() functions. For more information on getting/setting filter control parameters see Using Format Filters API for Filter Control.

The following TIFF filter control parameters are related to tiles:

- WRITE_CONFIG
- TILE_H_COUNT
- TILE_V_COUNT
- TILE_WIDTH
- TILE_HEIGHT
- BUFFER_SIZE

Please see the section TIFF in the File Format Reference for detailed information about the parameters.

The following example call shows an example image loaded into memory with only tiles 5, 6, 8, and 9, and then being saved as a tiled image with 20 tiles:

```
/* Declare a structure of type AT_STITCH */
AT_STITCH stitchStruct;
UINT nPage;
HIGEAR hIGear;
LONG lHorizTileCount, lVertTileCount;
/* Set uRefTile to the number of the tile that you would like to use as the upper-
left most tile in the stitched image */
stitchStruct.uRefTile = 5;
/* Set uTileRows to the number of tiles that you would like per row */
stitchStruct.uTileRows = 2;
/* Set uTileCols to the number of tiles that you would like per column */
stitchStruct.uTileCols = 2;
/* Set lHorizontalTileCount to the number of tiles per row that ImageGear should
make when an image is saved to disk as a tiled TIFF. */
lHorizTileCount = 5;
/* Set lVertTileCount to the number of rows that ImageGear should make when an image
is saved to disk as a tiled TIFF. */
lVertTileCount = 4;
/* Set nPage to the page number of a multi-page TIFF that you would like to load. If
the TIFF isn't multi-page, set to 1. */
nPage = 1;
/* This function loads the tiles specified by &stitchStruct, stitches them together
in a DIB and returns you a HIGEAR to the image in &lphIGear.*/
IG_load_tiles_stitch("KidsKatz.tif", nPage, &stitchStruct, &hIGear);
/* Use the Filter Control function to tell ImageGear how to configure TIFF files
when it saves them to disk. This call specifies that they should be saved as tiled
images and that you will be providing ImageGear will a fixed number of tiles to
create when saving the file. */
IG_fltr_ctrl_set(IG_FORMAT_TIF, "WRITE_CONFIG", (LPVOID)IG_TIF_TILED_FIXED_COUNT,
sizeof(lpWriteConfig));
/* Use the Filter Control function to tell ImageGear how many tiles per row you would
like to make in images saved to disk as TIFFs. */
IG_fltr_ctrl_set(IG_FORMAT_TIF, "TILE_H_COUNT", (LPVOID)lHorizTileCount,
sizeof(lHorizTileCount));
/* Use the Filter Control function to tell ImageGear how many rows of tiles you would
like to make in images saved to disk as TIFFs. */
IG_fltr_ctrl_set(IG_FORMAT_TIF, "TILE_V_COUNT", (LPVOID)lVertTileCount,
sizeof(lVertTileCount));
/* Call IG_fltr_save_file() to save the current HIGEAR image as an uncompressed
TIFF. */
IG_fltr_save_file(hIGear, "KidsKatz.tif", IG_SAVE_TIF_UNCOMP, 1, TRUE);
```

If you do not specify the configuration to use for saving a TIFF image to disk, ImageGear will use the default setting of IG_TIF_STRIP_FIXED_COUNT. If you do not specify a number of strips, the default of 1 is used.

TIFF Storage

The TIFF format filter supports four modes of storage for writing out to a TIFF file. All settings are made by supplying "WRITE_CONFIG" control parameter for the function IG_fltr_ctrl_set(). Here are the possible settings:

IG_TIF_STRIP_FIXED_COUNT	Write the image using a fixed number of strips. The number of strips to use can be set via the NUMBER_OF_STRIPS control parameter.
IG_TIF_STRIP_FIXED_BUFFER	Write the image using strips so that each strip is not greater than the specified size in bytes. The size of the strip buffer can be set via the BUFFER_SIZE control parameter. Please note that at least one raster will be included in the strip.
IG_TIF_TILED_FIXED_SIZE	Save the image using tiles of fixed size. The size of the tiles can be set via the TILE_WIDTH and TILE_HEIGHT control parameters.
IG_TIF_TILED_FIXED_COUNT	Save the image using a fixed number of tiles. The number of tiles in both the horizontal and vertical direction can be set via the control parameters TILE_H_COUNT and TILE_V_COUNT.

1.2.4.3.9 Internal Stream Bufferization

ImageGear uses internal bufferization for image reading and writing. This reduces the number of system IO operations during reading and writing, and consequently improves image loading and saving performance, especially when the image is located on a remote computer.

Use the IO.BUFFER_SIZE control parameter to control the size of the reading buffer. The default value is 262144 bytes (256 KBytes). This size works well for most images, but you can use a smaller buffer for very small images or a larger buffer for large images. Setting the buffer size to 0 cancels bufferization on reading.

Some format filters, such as TIFF and JPEG, set the buffer size for reading pixel data automatically, according to the image raster size. However, they also use the common IO.BUFFER_SIZE setting for reading the image header.

Many formats set the buffer size automatically.

1.2.4.4 Displaying Images

ImageGear's set of over 40 display-related functions allows you to control where and with what attributes your application displays each of its images. Among the attributes you can set on an image-by-image basis are:

- Contrast adjustment
- Brightness adjustment
- Gamma correction
- Dithering
- Anti-aliasing enhancement
- Rotation
- Region within the display area in which the image is to be displayed (this area is called the Device Rectangle)
- The portion of the image to display (this array of pixels within the DIB bitmap is called the Image Rectangle)
- How to fit the Image Rectangle to the Device Rectangle
- The background fill pattern and color to use (for any area of the Device Rectangle left vacant by the image)

Also, by using the image's LUTs (Look-Up-Tables), you can translate the colors in the image's color palette to other colors you select (and therefore you can, for example, display an 8-bit grayscale image in any 256 colors you choose).

All of the above display attributes take effect during display only. They do not alter either the image bitmap or the color palette in the DIB.

In addition, any time your image is being displayed, you can center, zoom, or scroll it from within your application.

This section provides information about how to use the features described above, and how to use additional special purpose IG_dspl_image_draw...() functions:

- Concepts
- Understanding Storage Options
- Understanding Display Options

1.2.4.4.1 Concepts

The key concepts of the new display functionality are display options, display option groups, and display operations.

- A display option is a variable that is assigned a value (or a setting) from a predefined list. Every option has a default value associated with it, and is therefore always defined.
- A display option group is the complete set of all options' values.
- A display operation is an action such as displaying or printing an image. Every operation requires an option group whose identifier is passed as a parameter to the function that performs the operation. This group affects the result of the operation and the way in which it is achieved.

1.2.4.4.2 Understanding Storage Options

Every handle of an ImageGear image (HIGEAR) contains a set of option groups. ImageGear does not impose any restrictions on the size of this set, i.e., on the number of option groups that are stored with any given HIGEAR. You do not need to allocate or de-allocate the storage space for the groups. You can, however, reset all group options to their default values.

Each option group has its own ID. This ID is a DWORD integer and is unique in the scope of a given HIGEAR; i.e., every pair {HIGEAR, DWORD} uniquely identifies an option group. The reason for introducing option groups and associating multiple groups with a single image handle is to provide a convenient means of drawing the same image on multiple devices. In this case, a separate option group may be allocated for each of the devices, and later used whenever the image is output to the corresponding device without the need to reassign options values.

1.2.4.4.3 Understanding Display Options

In this section, all the options are divided into several categories according to the type of the functionality they affect:

- Geometric Layout
- Dithering, Anti-Aliasing, and Palette Handling
- Transparency and Background
- Look-Up Tables and Gamma Correction
- Grayscale Look-Up Tables

1.2.4.4.3.1 Geometric Layout

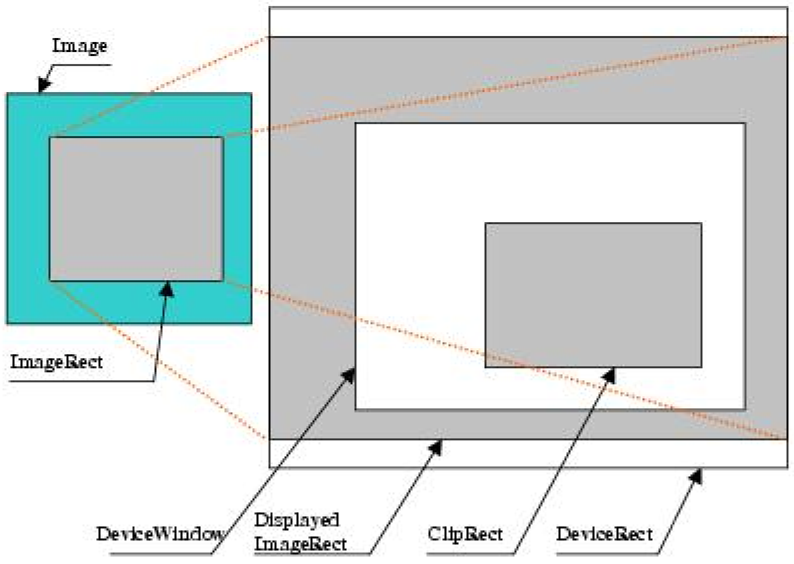
The Geometric Layout category includes options that determine the image layout on the destination device:

ImageRect (AT_RECTANGLE)	A rectangle that defines the part of the image that is output to the device. It is expressed in image coordinates. By default, this option is set to the entire image.
ClipRect (AT_RECTANGLE)	A rectangle that identifies the destination device area that is affected by display operations. Display operations never affect parts of the destination device outside of the ClipRect. It is expressed by device coordinates and is initially assigned the entire client area of the destination device.
DeviceRect (AT_RECTANGLE)	A rectangle that identifies the area on the destination device upon which the image is projected. Only the portion of the projection falling within the clip rectangle will be seen. The rest of the clip rectangle may be painted with the background color depending on BkMode. The image does not necessarily fit exactly into the device rectangle. The position of the image inside the device rectangle is determined by other options such as AspectMode, FitMode, and AlignMode (see their descriptions below).
DisplayedImageRect (AT_RECTANGLE)	This is not a display option that you can set, however this is a very important concept discussed below. This is the rectangular area that represents the image's size and location on the device (and therefore, it is expressed in device coordinates). It is important to understand that it is not related to ClipRect, which represents the visible part of the image. The DisplayedImageRect's value can be calculated and used but cannot be directly set.
AspectMode (AT_MODE) and AspectValue (DOUBLE)	These options determine an image's aspect ratio (its width-to-height ratio). AspectMode can be assigned one of two possible values: ● IG_DSPL_ASPECT_FIXED - this aspect ratio is the one contained in AspectValue. ● IG_DSPL_ASPECT_NOT_FIXED - this aspect ratio is that of the device rectangle. AspectValue may be any positive number with the following meaning: AspectValue = (DisplayedImageRect.Width / DisplayedImageRect.Height) / (ImageRect.Width / ImageRect.Height). The default values for these options are IG_DISPL_ASPECT_FIXED and 1.0, respectively.
PPMCorrect (BOOL)	This option allows you to take into account the image's resolution when calculating the aspect ratio, such that physical width and height are actually used for calculation. If this value is TRUE then ImageRect.Width and ImageRect.Height will be altered based on the horizontal and vertical resolution, respectively. The default value is FALSE.
FitMode (AT_MODE)	This value defines how an image fits to the device rectangle while preserving its aspect ratio according to the AspectMode and AspectValue options. The possible values are: ● IG_DSPL_FIT_TO_DEVICE - in this mode, the image is scaled to fit both the width and height of the device rectangle. ● IG_DSPL_FIT_TO_WIDTH - in this mode, the image is scaled to fit the width of the device rectangle. ● IG_DSPL_FIT_TO_HEIGHT - in this mode, the image is scaled to fit the height of the device rectangle. ● IG_DSPL_ACTUAL_SIZE - in this mode, the device rectangle is ignored, and the image is scaled 1:1. The default value is IG_DSPL_FIT_TO_DEVICE.
AlignMode (AT_MODE)	This value defines how the displayed image is aligned relative to the device rectangle. Possible values are bitwise ORs of any two values, such that one of them represents the horizontal alignment and the other represents the vertical alignment. The values representing the horizontal alignment are: ● IG_DSPL_ALIGN_X_LEFT - the image is aligned to the left border of the device rectangle. ● IG_DSPL_ALIGN_X_CENTER - the image is centered horizontally. ● IG_DSPL_ALIGN_X_RIGHT - the image is aligned to the right border of the device rectangle. The values representing the vertical alignment are: ● IG_DSPL_ALIGN_Y_TOP - the image is aligned to the top border of the device

	rectangle. • IG_DSPL_ALIGN_Y_CENTER - the image is centered vertically. • IG_DSPL_ALIGN_Y_BOTTOM - the image is aligned to the bottom border of the device rectangle. The default value is IG_DSPL_ALIGN_X_CENTER | IG_DSPL_ALIGN_Y_CENTER.
ZoomMode (AT_MODE)	The value that specifies how the image is zoomed in the horizontal and vertical directions. Possible values for this option are bitwise ORs of any two flags such that one of them represents a horizontal zoom value and the other represents a vertical zoom value. Flags representing the horizontal zoom values are: • IG_DSPL_ZOOM_H_NOT_FIXED - this mode allows any horizontal zoom value. • IG_DSPL_ZOOM_H_FIXED - in this mode, the horizontal zoom factor is taken from ZoomValueH (see below for details). Flags representing the vertical zoom values are: • IG_DSPL_ZOOM_V_NOT_FIXED - this mode allows any vertical zoom value. • IG_DSPL_ZOOM_V_FIXED - in this mode the vertical zoom factor is taken from ZoomValueV (see below for details). The default value for this option is IG_DSPL_ZOOM_H_NOT_FIXED | IG_DSPL_ZOOM_V_NOT_FIXED. Please note that AspectMode takes precedence over ZoomMode. In other words, the vertical zoom values are ignored if AspectMode is set to IG_DSPL_ASPECT_FIXED.
ZoomValueH, ZoomValueV (DOUBLE)	These options specify actual horizontal and vertical zoom values according to ZoomMode. Their meaning may be expressed as follows: ZoomValueH = DisplayedImageRect.width / ImageRect.width, ZoomValueV = DisplayedImageRect.height / ImageRect.height. Please note that ZoomValueV is not used if AspectMode is set to IG_DSPL_ASPECT_FIXED.
OrientMode (AT_MODE)	This parameter identifies how the image is oriented before it is drawn on the output device. Possible values are determined by the constants that have the form of IG_DSPL_ORIENT_X_Y, where each of X and Y can be LEFT, TOP, RIGHT, or BOTTOM. X stands for the position where the top-most row of the bitmap will be located after applying the transformation, and Y stands for the position where the left-most column of the bitmap will be located after applying the transformation. For example, IG_DSPL_ORIENT_RIGHT_TOP means that the left-most column will become the image's new top-most row, and the top-most row will become the image's new right-most column. The image will be rotated 90 degrees. The following constants are defined: • IG_DSPL_ORIENT_TOP_LEFT - the image is displayed unchanged. • IG_DSPL_ORIENT_LEFT_TOP - the image is rotated 270 degrees and then flipped vertically. • IG_DSPL_ORIENT_RIGHT_TOP - the image is rotated 90 degrees. • IG_DSPL_ORIENT_TOP_RIGHT - the image is flipped horizontally. • IG_DSPL_ORIENT_BOTTOM_RIGHT - the image is rotated 180 degrees. • IG_DSPL_ORIENT_RIGHT_BOTTOM - the image is rotated 90 degrees and then flipped vertically. • IG_DSPL_ORIENT_LEFT_BOTTOM - the image is rotated 270 degrees. • IG_DSPL_ORIENT_BOTTOM_LEFT - the image is flipped vertically. The default value is IG_DSPL_ORIENT_TOP_LEFT. Note that when changing an image's orientation, the image rectangle's orientation is also changed. The other rectangles and options remain unchanged.
ScrollbarMode (AT_MODE)	Scrolling is automatically supported in both horizontal and vertical directions. Scroll parameters are automatically calculated by ImageGear. All scroll parameters are stored apart from scrollbars, and therefore, the image can be scrolled in any possible way. If you want to manage the scrollbars yourself, you may do so by using this option. Possible values for this option include any combination of two flags, such that one of them is a "horizontal" flag and the other is a "vertical" flag. Horizontal flags are: IG_DSPL_HSCROLLBAR_AUTO - allows ImageGear to show and hide the horizontal scrollbar depending on the scroll range. IG_DSPL_HSCROLLBAR_ENABLE - in this

	mode, ImageGear always shows the horizontal scrollbar even if the scroll range is 0. IG_DSPL_HSCROLLBAR_DISABLE - in this mode, ImageGear ignores the horizontal scrollbar and does not set any of its properties. IG_DSPL_VSCROLLBAR_AUTO - allows ImageGear to show and hide the vertical scrollbar depending on the scroll range. IG_DSPL_VSCROLLBAR_ENABLE - in this mode, ImageGear always shows the vertical scrollbar even if the scroll range is 0. IG_DSPL_VSCROLLBAR_DISABLE - in this mode, ImageGear ignores the vertical scrollbar and does not set any of its properties.
ScrollPosH, ScrollPosV(LONG)	These options determine how DisplayedImageRect is moved in the horizontal and vertical directions. These values should fall within the corresponding scroll range. Both are set to 0 by default.
MapMode (DWORD), Viewport (AT_RECTANGLE),Window (AT_RECTANGLE)	These options are device dependent and allow you to set logical coordinates. It is assumed by ImageGear that all coordinates except ImageRect are logical. These options are necessary to properly convert logical to device coordinates. For the Windows platform the value of MapMode can be any value accepted by the GDI's functions GetMapMode/SetMapMode. The Viewport and Window contain values that are the same as GDI's SetViewportOrgEx/SetViewportExtEx and SetWindowOrgEx/SetWindowExtEx. The default value is the same as current desktop map mode.

This figure demonstrates the meaning of the rectangles listed above:



Some of the options listed above conflict unless priorities are defined. Consider the general algorithm of display rendering. It consists of several steps, and on each step ImageGear processes some options to introduce modifications to the resulting image.

- ImageRect is oriented according to OrientMode. ClipRect and DeviceRect are calculated according to their definitions above.
- If FitMode is set to IG_DSPL_ACTUAL_SIZE then DisplayedImageRect.width and DisplayImageRect.height are set to ImageRect.width and ImageRect.height respectively. Otherwise, DisplayedImageRect is computed using DeviceRect.
- According to ZoomMode, ZoomValueH and ZoomValueV, DisplayedImageRect.width and DisplayedImageRect.height are modified as follows:

```
if( (ZoomMode&IG_DSPL_ZOOM_H_FIXED) != 0 )
    DisplayedImageRect.width = DisplayedImageRect.width*ZoomValueH,
```

and

```
if( (ZoomMode&IG_DSPL_ZOOM_V_FIXED) != 0 )
    DisplayedImageRect.height = DisplayedImageRect.height*ZoomValueV.
```

If AspectMode is set to IG_DSPL_ASPECT_FIXED, then ZoomValueV is not used and the above procedure is changed in the following way:

```
If( (ZoomMode&IG_DSPL_ZOOM_H_FIXED) != 0 )
{
DisplayedImageRect.width = DisplayedImageRect.width*ZoomValueH,
DisplayedImageRect.height = DisplayedImageRect.height*ZoomValueH.
}
```

- DisplayedImageRect.x and DisplayedImageRect.y are computed so that DisplayedImageRect is aligned as specified by AlignMode.
- The scrolling range is calculated so that DisplayedImageRect can be viewed inside of ClipRect. Then DisplayedImageRect.x and DisplayedImageRect.y are shifted according to the current scroll position.

1.2.4.4.3.2 Dithering, Anti-Aliasing, and Palette Handling

ImageGear supports automatic dithering of source images. This functionality is affected by the following options:

DitherMode (AT_MODE)	Can be assigned the following values: ● IG_DSPL_DITHER_AUTO - Specifies that the destination device color resolution should be used for dithering. In this mode, ImageGear automatically applies dithering only when it is necessary. ● IG_DSPL_DITHER_TO_8BPP - This mode forces ImageGear to assume that the output device is 8 bits per pixel and perform the necessary dithering. ● IG_DSPL_DITHER_TO_4BPP - This mode forces ImageGear to assume that the output device is 4 bits per pixel and perform the necessary dithering. ● IG_DSPL_DITHER_TO_1BPP - This mode forces ImageGear to assume that the output device is 1 bit per pixel and perform the necessary dithering. ● IG_DSPL_DITHER_NONE - Disables ImageGear's dithering. In this mode, dithering is performed by the operating system or the device driver.

The following flags can be used with any of the above modes:

IG_DSPL_DITHER_FIXED_PALETTE	If this flag is set, ImageGear will try to use the standard palette when performing dithering. This may be useful if the output device contains more than one image and by using this flag it is possible to draw images with the same palette.
PALETTE	This flag is applicable only if the output device is 8 bits per pixel. It tells ImageGear to use the 216 entries Netscape palette. The default value for DitherMode is IG_DSPL_DITHER_AUTO. Currently only ordered dithering is implemented.
IG_DSPL_ANTIALIAS_MODE (AT_MODE)	A bitmask used to isolate black and white anti-aliasing modes: ● IG_DSPL_ANTIALIAS_NONE - Anti-aliasing is not used. ● IG_DSPL_ANTIALIAS_SCALE_TO_GRAY - The scale to gray algorithm is used, and the output image becomes 4 bits per pixel. ● IG_DSPL_ANTIALIAS_PRESERVE_BLACK - ImageGear will try to preserve black pixels while scaling. ● IG_DSPL_ANTIALIAS_PRESERVE_WHITE - ImageGear will try to preserve white pixels while scaling. Together with these modes, the following flag can be used: ● IG_DSPL_ANTIALIAS_SUBSAMPLE - ImageGear will use sub-sampling during anti-alias scaling. The output is as good, while the speed is greater.
IG_DSPL_ANTIALIAS_RESAMPLE_MODE	A bitmask used to isolate re-sampling modes: ● IG_DSPL_ANTIALIAS_RESAMPLE_BILINE - Resample with bilinear interpolation.
IG_DSPL_ANTIALIAS_COLOR_MASK	A bitmask used to isolate color anti-aliasing modes: ● IG_DSPL_ANTIALIAS_COLOR - Color anti-aliasing.
AliasThreshold (UINT)	Threshold integer value from 0 to 100. Its meaning depends on the AliasMode value. ● If AliasMode is set to IG_DSPL_ANTIALIAS_SCALE_TO_GRAY, then AliasThreshold determines how many black and white pixels are involved in the destination gray pixel value. A value of 100 causes ImageGear to take 100% white pixels; a value of 0 causes ImageGear to take 100% of black pixels; and the default value is 50, which means 50% of white and 50% of black pixels. ● If AliasMode is set to IG_DSPL_ANTIALIAS_PRESERVE_BLACK, then AliasThreshold determines how many black pixels should be preserved. A value of 100 means that 100% of black pixels are preserved. The default value is 50. ● If AliasMode is set to IG_DSPL_ANTIALIAS_PRESERVE_WHITE, then AliasThreshold determines how many white pixels should be preserved. A value of 100 means that 100% of white pixels are preserved. The default value is 50.

DevicePalette (HPALETTE)	This option is a palette in OS-dependent format. This palette is implemented before drawing pixels onto the output device. By default, this option is NULL, which means that it is created every time; either from the logical palette of the image, or from the logical palette used in dithering. If this option is not NULL, then it should be a valid palette handle. ImageGear does not perform logical processing of the specified palette; it only implements it before drawing. Incorrect usage of this option may distort the image on the destination device.
PaletteMode (AT_MODE)	This option specifies how to use the DevicePalette option when the destination device does support palette operations. Possible values are: • IG_DSPL_PALETTE_HIGH - ImageGear will use the palette in the high priority mode. This means that the operating system palette manager will try to best map colors of DevicePalette to the system palette. • IG_DSPL_PALETTE_LOW - ImageGear will use the palette in the low priority mode. In this mode, the palette manager will try to best preserve the current view of the destination while drawing the new image on it. • IG_DSPL_PALETTE_DISABLE - ImageGear will not implement DevicePalette in the destination device while drawing the image.

1.2.4.4.3.3 Transparency and Background

The ImageGear display functionality supports transparency by color and by mask. The former means that it is possible to draw images with the specified color being transparent. The latter allows using a specified 1-bit per pixel image as the transparent mask. There are a few options related to transparency and background support:

TranspMode (AT_MODE)	Option that contains transparency flags. If its value is IG_DSPL_TRANSPARENCY_NONE then transparency is disabled. Each of the following flags enables some transparency-related features: • IG_DSPL_TRANSPARENCY_COLOR - If this flag is set, then the transparent color is enabled, and the color's value that is assigned to the TranspColor option is used as transparent while drawing the image. • IG_DSPL_TRANSPARENCY_MASK - If this flag is set, then the transparency mask is enabled, and the TranspMask option is used as the transparent bitmap. • IG_DSPL_TRANSPMASK_STRETCH_TO_IMAGE - This flag is used when the transparency mask is enabled. If this flag is set, then the TranspMask image is resized and oriented along with the image being displayed. In other words, if during the display operation the image is scaled by factors DX and DY in the horizontal and vertical directions respectively, then the TranspMask image is scaled by the same factors. If this flag is not set then the transparency mask is not scaled. This flag is ignored unless the IG_DSPL_TRANSPMASK_LOCATE_TO_IMAGE flag is set. The next three flags are exclusive; only one of them can be set at a time: • IG_DSPL_TRANSPMASK_LOCATE_TO_IMAGE - This flag is applicable if the transparency mask is enabled. If this flag is set, then the mask is calculated in the image-dependent coordinate system. This means that the mask is oriented as the image is, and the MaskLocation option is calculated from the original image's ImageRect. If this flag is not set, then the IG_DSPL_TRANSPMASK_STRETCH_TO_IMAGE flag is ignored. • IG_DSPL_TRANSPMASK_LOCATE_TO_CLIPRECT - This flag locates the transparent mask relative to the ClipRect, so that the MaskLocation is calculated from this rectangle's left top point. • IG_DSPL_TRANSPMASK_LOCATE_ABSOLUTE - This flag locates the transparent mask on the output device so that the mask's left top point has the coordinates specified in the MaskLocation option. The default value for TranspMode option is IG_DSPL_TRANSPARENCY_NONE, which means that transparency is disabled.
TranspColor (AT_RGB)	This (RGB) triple specifies the transparent color, and it is used if the IG_DSPL_TRANSPARENCY_COLOR flag is set in TranspMode (see its description above). If the transparent color is enabled, then all the pixels in the image that have values equal to TranspColor are drawn as transparent.
TranspMask (HIGEAR)	This option specifies the transparency mask, and it is used if the IG_DSPL_TRANSPARENCY_MASK flag is set (see the TranspMode's description above). This is a normal HIGEAR image, and its location on the screen depends on other flags described above and the MaskLocation option. The application code is responsible for creating and deleting this mask.
MaskRect (AT_RECTANGLE)	This rectangle specifies which part of the TranspMask image should be used as the transparency mask. It is an analog of the original image's ImageRect, but it applies to the TranspMask image. By default, it is initialized with the empty rectangle, which means the complete TranspMask image will be used.
MaskLocation (AT_POINT)	This option specifies how the transparency mask is located relative to either the image or the device, depending on the flags set in TranspMode.
BkMode (AT_MODE)	This option specifies how ImageGear should fill the area of ClientRect that is not covered by the image's pixels. If its value is IG_DSPL_BACKGROUND_NONE, then the background is disabled, and ImageGear does not fill this area. Other possible values are as follows: • IG_DSPL_BACKGROUND_UNDER_IMAGE - If this flag is set, then the image's transparent pixels are drawn with current BkColor and BkBrush (see their description below). The area outside of DisplayedImageRect is not affected. • IG_DSPL_BACKGROUND_BEYOND_IMAGE - If this flag is set, then the transparent pixels that are outside of DisplayedImageRect are drawn with current BkColor and BkBrush (see their description below). The default value for this option is IG_DSPL_BACKGROUND_UNDER_IMAGE|

	IG_DSPL_BACKGROUND_BEYOND_IMAGE.
BkColor (AT_RGB)	This option is a (RGB) triple that specifies the background color. This color is used to fill the area of ClipRect that is not covered by the image's pixels.
BkBrush (HBITMAP)	This option is the handle of the bitmap that stores the brush to be used. The application code is responsible for creating and deleting this brush.

1.2.4.4.3.4 Look-Up Tables and Gamma Correction

ImageGear supports gamma correction preprocessing before the image is drawn onto the destination device. This operation does not affect the image itself, but only changes its appearance.

There are three options that allow you to control the gamma correction:

- RedLut (LPBYTE) - this is a 256-entry array of bytes that contains a new value for each of the possible 256 intensities of the red color.
- GreenLut (LPBYTE) - this is a 256-entry array of bytes that contains a new value for each of the possible 256 intensities of the green color.
- BlueLut (LPBYTE)- this is a 256-entry array of bytes that contains a new values for each of the possible 256 intensities of the blue color.

The default value for each of the three options is the identity array.

If the source image is not in the RGB color space, then it is first converted to the RGB color space, and then all the look-up tables are applied.

The following are high-level options that automatically create all necessary look-up tables:

- Contrast, Brightness, Gamma (DOUBLE) - these three options are actually parameters that specify how to calculate the look-up tables to get the necessary color effects.
- Parameter Contrast - specifies the contrast level to produce. Values greater than 1.0 increase contrast; values less than 1.0 decrease contrast. Values less than 0.0 will invert contrast (exchange dark and light).
- Parameter Brightness - specifies the brightness adjustment. Possible values range from -255.0 to +255.0.
- Parameter Gamma - controls the non-linear contrast adjustment. Values greater than 1.0 increase contrast; values less than 1.0 decrease it. Usual range is from 1.8 to 2.2

The default values are:

- Contrast = 0.0,
- Brightness = 1.0,
- Gamma = 1.0

1.2.4.4.3.5 Grayscale Look-Up Tables

The ImageGear display API allows storing a grayscale (single-channel) look-up table (LUT) with an image's display settings. The grayscale look-up table can be set for any image, but ImageGear only uses it with 8...16-bit grayscale images, and ignores it with the other images. This look-up table specifies a transform from 16-bit image to 8-bit image, which allows the display of a particular part of an 8...16-bit image's contrast range, or enhanced image's contrast.

A grayscale LUT can be stored with a 16g image as well. However, if both the image and its display contain LUTs, the display LUT overrides the image's LUT. A LUT can be removed from an image and attached to a display, or vice versa.

Storing a grayscale LUT with display settings allows you to display the same image with different LUTs simultaneously in different windows.

A grayscale LUT can be used in combination with a RGB LUT. The grayscale LUT is applied first, and then the RGB LUT is applied.

Use IG_dspl_grayscale_LUT_update_from() to create or update a grayscale LUT with the specified LUT. LUT data will be copied to the display settings. Set the lut parameter to NULL to remove the grayscale LUT from display settings.

Use IG_dspl_grayscale_LUT_exists() to check whether display settings contain a grayscale LUT.

Use IG_dspl_grayscale_LUT_copy_get() to obtain a copy of the display grayscale LUT.

The only allowed LUT configuration for display is: InputDepth = 16, OutputDepth = 8, Output is unsigned. Such LUTs can be used with grayscale images whose depth is 8... 16 bits per pixel.

See Also:

Working with Grayscale Look-Up Tables

Displaying Medical Grayscale Images

1.2.4.5 Printing Images

ImageGear provides a simple all-purpose printing function that will print any image to a graphics-capable printer. This function is:

```
IG_dspl_image_print ( HIGEAR hIGear, DWORD dwGrpID, HDC hDC, BOOL bDirectToDriver );
```

This function prints a HIGEAR image to the current default printer according to the display parameter specified by dwGrpID group. There is a special group IG_GRP_DEFAULT_PRINT that can be used to print an image with the default print options.

- When bDirectToDriver = TRUE, ImageGear sends the image's DIB directly to the printer's device driver. In this case, the entire procedure is controlled by the printer's driver. If your printer has special capabilities such as color, and if the driver supports these, then your image can be printed with these features.
- When bDirectToDriver = FALSE, ImageGear handles the printing procedure as follows:
  - ImageGear first reduces the image to 1-bit, if necessary (such as by using a Bayer dithering algorithm).
  - It then sends each raster line to the printer driver individually.
  - If you've called IG_status_bar_CB_register() to declare a status bar callback function, ImageGear calls your callback function after each raster line is sent. This type of callback permits you to display a status bar showing the completed percentage, or a message box displaying the page number being sent. You can also detect a keystroke or mouse selection indicating that the user wants to cancel the printing process.
  - The sample application program "print.c" demonstrates how to implement these and other features, including an initial Print Dialog Box.

In general, bDirectToDriver = FALSE gives you greater control of the printing process, while bDirectToDriver = TRUE gives you faster printing.

Note also that the functions IG_dspl_page_print(), IG_dspl_document_print(), and IG_dspl_document_print_custom() allow you to specify how to print a single image on a page, and how to print a list of images on a page, specifying how to place the images relative to the page's borders.

1.2.4.6 Processing Images

ImageGear's comprehensive family of image processing functions permits you to perform both simple and complex image-modifying operations using a single function call. Image alterations such as contrast enhancement, sharpness adjustment, color reduction/promotion, image merging ("blending"), and "special effects" are performed using the functions in this group.

Image processing functions are always named beginning with IG_IP_ ...() or IG_FX_ ...(). Be careful not to confuse these functions with IG_dspl_image_draw ...() functions designed to perform similar operations while displaying images. The IG_dspl_image_draw ...() functions affect only how the image will be displayed. The image processing functions actually alter the pixel data in the DIB image bitmap, or the DIB palette, or both.

Many image processing functions permit you to specify a rectangular region within your image, limiting the function's operation to that region. In such cases, the address of an AT_RECT structure is supplied as an argument (specify this argument as NULL to operate on the entire image).

The "image rectangle" setting made by calls to function IG_dspl_layout_set() is not used by the image processing functions.

In this section, ImageGear's image processing functions are grouped as shown below. You may want to refer to the individual discussions of the groups as they become pertinent to your application's development needs.

- Geometric Transformations
- Contrast Alteration
- Color Reduction
- Color Promotion
- Blending and Combining Images
- Image Correction
- Image Encryption
- Image Analysis
- Region of Interest Processing

If you encounter an occasional image processing term with which you are not familiar, be sure to refer to the Glossary. Also, refer to the function entries in the Core Component API Function Reference for the detailed calling sequences and additional information and examples. In a function's entry, be sure to check the "Bits Per Pixel" line. It specifies which bit depths may be processed using that function. The descriptions of the different bit depths (1, 4, 8i, 8-bit gray level, 9-16-bit gray level, 24, and 32) and how they are stored internally in the DIB image bitmap in memory, can be found in Understanding Bitmap Images.

A number of image processing functions can process 8-bit gray level and 24-bit images, but cannot process 8i (8-bit indexed color) images. This is because in 8i images the pixel value does not itself describe the pixel (its color or its intensity), but is merely an index into the palette. If you want to call such a function for an 8i image, first promote the image to 24-bit using function IG_IP_color_promote().

1.2.4.6.1 Geometric Transformations

An image transformation function is one in which a mathematical algorithm is applied to transform each pixel to a new location or value. Rotation is a good example of a simple image transformation. Image transformation differs from contrast adjustment (which might also be carried out by applying an algorithm) in that the contrast adjustment is trying to achieve some visual improvement or enhancement of the image. A transformation's object is simply to achieve the transformation (e.g., rotate the image).

ImageGear provides the following image transformation functions. Refer to the description of each in Core Component API Function Reference for detailed calling sequences as well as other related information:

IG_IP_convolve_matrix()	Convolves the 8-bit gray level or 24-bit image using a user-defined convolution kernel.
IG_IP_flip()	Flips your image right-for-left or top-for-bottom. This is equivalent to rotating the image around a vertical or horizontal axis (respectively) drawn through its center.
IG_IP_resize()	Re-scales the image, changing the size of the image bitmap.
IG_IP_rotate_any_angle()	Rotates your image through any angle you specify around its center.
IG_IP_rotate_multiple_90()	Rotates your image 90, 180, or 270 degrees around its center.
IG_IP_rotate_multiple_90_opt()	Rotates the image referenced by hIGear at an angle that is a multiple of 90 degrees, using additional rotation options.
IG_IP_sharpen()	Causes the dark side of a contrast boundary to become darker and the bright side to become brighter. This makes the image appear sharper. You can control the degree of sharpening applied.
IG_IP_smooth()	Removes graininess in an image, tending to soften or smooth its appearance. You control the degree of smoothing.
IG_IP_transform_with_LUT()	Transforms an image by mapping each pixel value through a Look-Up Table that you supply to obtain the pixel's new value.

1.2.4.6.2 Contrast Alteration

ImageGear's contrast alteration functions operate by altering the range of pixel intensities that occur in your image, or by redistributing the occurrence frequency of the pixel intensities. There are five IG_IP_contrast_...() functions, which operate as follows:

IG_IP_contrast_adjust()	Adjusts the contrast of the image by stretching or compressing the range of intensities that occur. Also adjusts brightness by adding or subtracting the specified constant to each intensity value.
IG_IP_contrast_stretch()	Adjusts the contrast of the image by stretching the range of intensities that occur, such that the least intense pixel becomes full black, and the most intense becomes full white.
IG_IP_contrast_equalize()	Adjusts the contrast of the image by stretching or compressing sub-ranges of intensities that occur, so that there are an approximately equal number of pixels in each sub-range. This can bring out subtle changes in contrast when contrast is poor in the original, such as in x-ray images.
IG_IP_contrast_gamma()	Adjusts the contrast of the image non-linearly, using an algorithm that tends to correct for the non-linear response of display monitor phosphors, video camera photoreceptors, and photographic emulsions.
IG_IP_contrast_invert()	Inverts each pixel intensity or color, resulting in an image that is a "negative" of the original.

Each of the above functions can operate on any specified rectangular portion of your image. However, when operating on an entire image, each function above can achieve its effect by altering the image's palette instead of by altering its pixel values. An example of a call to an IG_IP_contrast_ ...() function is:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_RECT rcRect; /* rectangle to operate on */IG_IP_contrast_equalize ( hIGear, &rcRect,
IG_CONTRAST_PIXEL);
```

If you specify IG_CONTRAST_PALETTE instead of IG_CONTRAST_PIXEL, note that your rectangle argument will be ignored, and the operation will be performed on the entire image.

Several additional functions help you to highlight contrast boundaries:

IG_IP_edge_map()	Produces an image that shows where there are contrast changes in the original image. An area in which there are no contrast changes is black in the resultant image; the stronger the contrast change, the brighter the result. The result tends to have bright lines where there are sharp contrast changes.
IG_IP_pseudocolor_small_grads()	Colors an 8-bit gray level image according to the local rate of pixel value change. This can be set to expose even very small gradients in brightness.
IG_IP_pseudocolor_limits()	Allows you to block out, to a single color, all pixels below (or above) a given pixel value. This can be used to highlight the portion of an image that is relevant (then IG_IP_contrast_stretch() might be called to enhance its contrast), or may be used to see what portions of the image are saturated or unsaturated.

Refer to the descriptions of all of the above in Core Component API Function Reference.

1.2.4.6.3 Color Reduction

Color reduction in general results in an image with fewer colors than the original, even if the bit depth is not changed. There are a number of color reduction methods, and ImageGear provides several different functions from which you can choose the one that is most suitable in a given case. Below you can see the color reduction methods that are available, and the functions that perform them:

IG_IP_color_reduce_bayer()	Reduces a 4, 8, or 24-bit image to a 1-bit or 4-bit image using a Bayer dithering algorithm.
IG_IP_color_reduce_to_bitonal()	Reduces a 4, 8, or 24-bit image to a 1-bit or "bi-tonal" image.
IG_IP_color_reduce_median_cut()	Reduces a 24-bit image to an 8-bit image using the median cut algorithm.
IG_IP_color_reduce_diffuse()	Reduces a 4, 8, or 24-bit image to a 1-bit or 4-bit image using a diffusion algorithm.
IG_IP_color_reduce_popularity()	Reduces a 24-bit image to an 8-bit image, while preserving its most prevalent or popular colors.
IG_IP_color_reduce_octree()	Reduces a 24-bit or 8-bit image to an 8-bit or 4-bit image. Uses an efficient algorithm that gives a result as close to the original as possible, using the number of colors you specify.
IG_IP_color_reduce_halftone()	Reduces a 4, 8, or 24-bit image to a 1-bit image using a halftone pattern.

Refer to the function descriptions in the Core Component API Function Reference for the calling sequences and further information on the above.

The additional function IG_IP_convert_to_gray() is also considered a color reduction function. It is called as follows:

```
IG_IP_convert_to_gray ( hIGear );
```

It always converts the image to 8-bit gray level. The resulting DIB has a 256-entry palette, and each 8-bit pixel value in the resulting image bitmap is a weighted average of the three color intensities of the pixel in the original image.

The following table lists supported input and output bit depths for color reduction functions:

Function Name	**Bpp In**	**Use or create palette for output image**	**BppOut**
IG_IP_color_reduce_bayer()	24	use standard B/W palette	1
	8	use standard B/W palette	1
	4	use standard B/W palette	1
	24	use given palette	4
	8	use given palette	4
	24	use standard palette	4
	8	use standard palette	4
IG_IP_color_reduce_diffuse()	24	use standard B/W palette	1
	8	use standard B/W palette	1
	4	use standard B/W palette	1
	24	use given palette	4
	8	use given palette	4
	24	use standard palette	4
	8	use standard palette	4
IG_IP_color_reduce_halftone()	24	use standard B/W palette	1
	8	use standard B/W palette	1
	4	use standard B/W palette	1
IG_IP_color_reduce_median_cut()	24	create optimal palette	8

IG_IP_color_reduce_octree()	24	use given palette (maxcolors > 16)	8
	8	use given palette (maxcolors > 16)	8
	24	create optimal palette (maxcolors > 16)	8
	8	create optimal palette (maxcolors > 16)	8
	24	use given palette (maxcolors <= 16)	4
	8	use given palette (maxcolors <= 16)	4
	24	create optimal palette (maxcolors <= 16)	4
	8	create optimal palette (maxcolors <= 16)	4
IG_IP_color_reduce_popularity()	24	create optimal palette	8
IG_IP_color_reduce_to_bitonal()	24	use standard B/W palette	1
	8	use standard B/W palette	1
	4	use standard B/W palette	1

1.2.4.6.4 Color Promotion

Color promotion is the process of increasing the bit depth, or number of bits per pixel, of an image. The color of each pixel is retained. ImageGear provides one function that handles promotion to any bit depth. It is called as follows:

```
IG_IP_color_promote ( hIGear, IG_PROMOTE_TO_24 );
```

The constant shown in the example above may instead be IG_PROMOTE_TO_4 or IG_PROMOTE_TO_8. Your image must originally have fewer bits per pixel than the bit depth to which you are promoting it.

When promoting to 4 or 8 bits, the promotion is accomplished by simply increasing the number of bits per pixel for each pixel (but without changing the pixel's value), and increasing the size of the DIB palette (the added palette entries are each set to zero).

When promoting to 24 bits, the 24-bit color of the pixel (obtained from the image's original palette) becomes the 24-bit pixel in the resulting promoted image. The DIB palette is deleted, since a 24-bit image does not have a DIB palette.

The function IG_IP_convert_to_gray(), though it could increase the number of bits per pixel in an image, is considered a color reduction function, because it reduces the colors to shades of gray.

1.2.4.6.5 Blending and Combining Images

A number of ImageGear's image processing functions combine data from two or more HIGEAR images, altering one with the result that it is a "blend" or other combination of the original images. These functions invariably take at least two HIGEAR handles as arguments. The functions in this group include:

IG_IP_blend_with_LUT()	Performs weighted blend of image 2 into image 1, with the weighting factor determined by looking up the pixel/color value in your LUT.
IG_FX_chroma_key()	This Special Effects function replaces image 1 by image 2 only where the image 1 pixels are within a specified hue range.
IG_IP_color_combine_ex()	Combines the pixel values from separate 8-bit images to form a single 24-bit color image.
IG_IP_color_separate()	The opposite of IG_IP_color_combine_ex(). Separates a 24-bit image's Red, Green, and Blue color values, producing separate 8-bit grayscale images (each with their own HIGEAR handle).
IG_IP_blend_percent()	Blends together the pixel values (if grayscale) or color values (if 24-bit) of the two images, according to a percent parameter that you specify. You can specify, for example, to combine 80% of image 1's value, with 20% of image 2's value. For a 24-bit image, you can specify that only one color channel (Red, Green, or Blue), or that all three color channels, are to be blended.

When you use IG_IP_blend_with_LUT(), you provide the address of a 256-byte Look-Up Table (LUT). ImageGear looks up image 1's pixel value or color value in your LUT to determine the percentages to blend for that pixel.

A different kind of blend is provided by Special Effects function IG_FX_chroma_key(). This function searches for pixels in image 1 that are within a color hue range that you specify. Only pixels within that color hue range are replaced from image 2. Arguments are provided for you to control the smoothness of the transition where replacement occurs, and a threshold pixel darkness below which replacement will not occur. You might use this to remove a person's picture from a plain background and to replace the original background with a new one. TV stations often use this technique to overlay the weather reporter onto a background of a weather map.

IG_IP_color_combine_ex() and IG_IP_color_separate() let you, respectively, assemble a 24-bit image from separate 8-bit gray level images, or disassemble a 24-bit image into separate 8-bit gray level images. In both cases, you can specify whether the 24-bit combination is standard RGB, or whether or not the 24-bit value is to be interpreted in terms of a different color space scheme. You may specify IG_COLOR_SPACE_RGB, or IG_COLOR_SPACE_IHS, or others. The complete list of supported color space schemes is provided in the file accucnst.h.

1.2.4.6.6 Image Correction

ImageGear provides several functions whose specific purpose is to correct an image's appearance. The functions in this group include:

IG_IP_despeckle()	Removes noise from a 1-bit image, using a 3x3 median filter on the image.
IG_IP_median()	Uses a median filter to reduce noise in 8-bit gray level and 24-bit images.
IG_IP_deskew_auto()	De-skews a 1-bit image that was created by reading a text file. You can specify a threshold angle below which de-skewing should not be performed.
IG_IP_deskew_angle_find()	Will analyze a 1-bit image and report to you the angle at which it is skewed.

In general, you would use the image correction functions with images that have been obtained by scanning text documents or graphical images.

1.2.4.6.7 Image Encryption

ImageGear provides a pair of image processing functions that perform image encryption/decryption operations. These functions change the image bitmap, modifying the format of the pixel data stored within it.

IG_IP_encrypt()	Encrypts the image bitmap using the specified password string. To successfully decrypt the image bitmap, the specified password will be needed.
IG_IP_decrypt()	Decrypts an image bitmap using the password string specified during encryption. If the password is the same as used with IG_IP_encrypt(), then the decryption will be successful, and the original image bitmap will be restored.

1.2.4.6.8 Image Analysis

This section describes the following:

- Histogram
- Color Counting

Histogram

ImageGear provides a histogram-generating function (and a related function to clear histogram bins) to assist you in image analysis. For an 8-bit image, whether 8i or 8-bit gray level, these functions can be called as follows:

```
HIGEAR hIGear; /* HIGEAR handle of image */
DWORD dwHistoBins[256]; /* Array of bins for counting */IG_IP_histo_clear( &dwHistoBins, 256
); /* Clear the bins */IG_IP_histo_tabulate (hIGear, &dwHistoBins, 256, NULL, 1, 0 );
/* Tabulate: */
```

In the above call, 256 DWORD bins are provided (one for each possible pixel value that can occur). The function will examine each pixel in the image bitmap, and will increment the bin corresponding to that value; that is, the bin dwHistoBins[pixel value] will be incremented. Upon return, you will have a count of the number of occurrences of each pixel value.

The fourth argument in the above call lets you specify the address of a rectangular region if you want to restrict the tabulating to include only a portion of the image. When NULL, the whole image is included in the tabulation.

The fifth argument above can be set higher than 1 if you want to increase the speed of the tabulation for a large image. 1 means that every raster line of pixels will be included in the tabulation. A higher value would result in raster lines being skipped.

The final argument in a call to IG_IP_histo_tabulate() is relevant for 24-bit images only. Its use is shown in the following example:

```
HIGEAR hIGear; /* Handle of a 24-bit image */
DWORD dwHistoBins[256]; /* Array of bins for counting */
IG_IP_histo_clear ( &dwHistoBins, 256 );/* Clear the bins */
IG_IP_histo_tabulate (hIGear, &dwHistoBins, 256, NULL, 1,
IG_COLOR_COMP_R ); /* Tabulate:   */
```

For 24-bit images, only 1 color channel is tabulated by a given call. That is, only 1 byte of each 3-byte pixel is examined. The final argument tells which byte (Blue, Green, or Red) should be tabulated.

You can also call IG_IP_histo_tabulate() for 1-bit and 4-bit images. In these cases, the number of DWORD histogram bins you need would be 2 and 16, respectively. In any event, always remember to call IG_IP_histo_clear() before calling IG_IP_histo_tabulate(), unless it is your intention to accumulate the count into the existing contents of your bins.

Color Counting

The IG_IP_color_count_get() function counts the number of different colors in the specified rectangle of an image.

1.2.4.6.9 Region of Interest Processing

This group of functions allows you to apply image processing and special effects functions on an arbitrary region of interest (ROI). You can specify a shape, such as ellipse, polygon, or freehand, or a 1-bit mask for identifying which pixels to include/exclude from image processing algorithms.

To apply an image processing function on a non-rectangular ROI, create the ROI using IG_IP_NR_ROI_to_HIGEAR_mask(), or use a 1-bit image, and associate it with the HIGEAR handle on which you are going to apply the processing function, using IG_IP_NR_ROI_mask_associate(). Consequent calls to image processing functions on this image will only affect the area specified by the ROI.

IG_IP_NR_ROI_control_get()	Returns the current ROI control settings.
IG_IP_NR_ROI_control_set()	Detects an area of pixels using a specified threshold.
IG_IP_NR_ROI_mask_associate()	Associates a mask HIGEAR, as specified by the AT_NR_ROI_MASK structure, with the image referenced by hIGear.
IG_IP_NR_ROI_mask_delete()	Deletes the mask HIGEAR created by IG_IP_NR_ROI_to_HIGEAR_mask().
IG_IP_NR_ROI_mask_unassociate()	Removes the non-rectangular ROI information from a HIGEAR image, but does not delete the mask HIGEAR.
IG_IP_NR_ROI_to_HIGEAR_mask()	Builds a non-rectangular ROI mask from a set of segment descriptors that you pass in.

1.2.4.7 Color Management

This section provides information about the following:

- Using Color Profile Manager
  - Color Profile Basic Concepts
  - ImageGear Color Profile Groups
  - Color Profile Manager API

1.2.4.7.1 Using Color Profile Manager

ImageGear supports color profiles and can perform color conversion operations. Detailed information about color processing based on profiles can be found at www.color.org.

The following sections describe how color profiles can be used with ImageGear:

- Color Profile Basic Concepts
- ImageGear Color Profile Groups
- Color Profile Manager API

1.2.4.7.1.1 Color Profile Basic Concepts

The first step in the color profile process is loading raster data from an external location (file, memory, or other) and converting it from some graphical file format to an internal uniform format that is incorporated into the HIGEAR object. After the image is loaded into memory, it is possible to perform different manipulations with it such as image processing transforms, color conversions, displaying, printing, and, finally, an export operation that converts raster data from internal representation into an external graphics file format.

All these steps may require color conversions. As the color profile is always associated with the appropriate device, there are several virtual devices defined in ImageGear. These "virtual devices" are not real devices, but abstract things that have associated color spaces and are used to convert color data from one color space to another. There are three virtual devices defined in ImageGear:

- Import devices - used for all import operations. Such import operations as loading a raster image using the format filter assume that imported color data is dependent on the import device. In other words, any format filter that loads color data in a device-dependent format from an external file format assumes that this data is dependent on this specific device.
- Export devices - the same as the import device, but used for export operations.
- Working devices - associated with color data stored in a HIGEAR object.

So we can assume every color conversion is a transition from one device to another. For example, loading an image from an external file using the file format filter (import device) to the internal HIGEAR representation (working device) assumes that the color data has to be "copied" from the import to the working device, i.e., color data has to be converted using color profiles associated with import and working devices.

ImageGear supports different color spaces such as RGB, CMYK, and grayscale. So each virtual device may be used with different color spaces. For example, the format filter may load pixels from an external file format in CMYK color space, and it will be necessary to convert color data from the CMYK color space associated with the import device to CMYK or another color space associated with the working device. Each device may have associated color profiles for different color spaces, and they are organized into groups - one group for each device.

You can find the exact definition of the term "color profile" in public specification ICC.1:1998-09 or newer. In general a color profile consists of a set of objects and transforms the specified parameters of color conversion from a standard device-independent color space (PCS) to a necessary device-dependent one. The ImageGear color management system accepts a color profile in the format specified in the ICC.1:1998-09 specification and converts it into internal representation for faster processing.

1.2.4.7.1.2 ImageGear Color Profile Groups

ImageGear allows you to set and get the actual value of a color profile in every group. In spite of the fact that operations with color profiles from different groups are very similar, at a low level there are some differences in how profiles from different groups are processed.

ImageGear allocates 3 color profile groups:

- ICP (Import Color Profile): This group of profiles is used during a filter load operation. In some cases a raster image file contains device-dependent color data such as RGB or CMYK but does not have a color profile associated with it. So this color data can be interpreted in a different manner depending on the color profile used. The ICP profile of appropriate color space should be used in this case. However, if a raster image contains a color profile, this one is used instead of the standard ICP profile.
- ECP (Export Color Profile): This group of profiles is very similar to ICP but is used in the filter export operation. If the raster image file to be exported does not allow you to store the color profile or if it needs to be stored in a specific color space, ECP allows you to provide the profile, and the output image will be converted to that color space.
- WCP (Working Color Profile): This group of color profiles provides information about the default color global parameters used to represent the color data for HIGEAR objects. Those global parameters are used if the image does not have a local color profile associated with it.

When color processing is performed on an image, either a local or global color profile is used. When an image is exported or imported, its pixel data, converted from one color space to another, is described by the WCP associated with the image and corresponding profile from the ECP or ICP group.

1.2.4.7.1.3 Color Profile Manager API

There are public functions implemented in ImageGear that allow you to set and get actual values for each group of color profiles.

To start working with the color profile manager, it is necessary to activate it. By default, the color profile manager is disabled in ImageGear, and all color-related operations are exactly the same as in previous versions. There is a Boolean global parameter named CPM.ENABLE_PROFILES used to control color profile management.

To activate color profile management, call the ImageGear global control function (see also Working with Global Control Parameters):

```
AT_BOOL     bEnable = TRUE;IG_gctrl_item_set( "CPM.ENABLE_PROFILES", AM_TID_AT_BOOL,
&bEnable,
sizeof(AT_BOOL), NULL );
```

When using color management, additional files are necessary: color profile files to be used as default profiles. By default those files should be located in the same folder as ImageGear and its components, and those files should be named as "ig_rgb_profile.icm" and "ig_cmyk_profile.icm" for RGB and CMYK color spaces, respectively. Correspondent global parameters CPM.RGB_PROFILE_PATH and CPM.CMYK_PROFILE_PATH exist to specify the full path to default profiles. Their values can be changed using the function IG_gctrl_item_set(), like in the following example:

```
CHAR     profile[256];
strcpy( profile, "d:\\profile\\rgb_profile.icm" );
IG_gctrl_item_set( "CPM.RGB_PROFILE_PATH", AM_TID_MAKELP(AM_TID_CHAR), profile,
strlen(profile)+1, NULL );
```

After color profile management is activated, it is possible to use the next functions to set color profiles in global parameters or to a specific image and to get information about currently used profiles.

The function:

```
IG_cpm_profile_set(       AT_MODE nColorSpace,        DWORD nProfileGroup,        LPBYTE
lpRawData,         DWORD dwRawSize,        AT_BOOL  bConvert);
```

allows you to set a new color profile value (lpRawData) to a group specified by the second argument and associate it with the color space given by the first argument. lpRawData is a pointer to the memory buffer that contains the color profiles in valid ICC format, and if it is NULL then the default color profile will be set. The last argument specifies how to process the color data of the images associated with it. If it is TRUE then all images associated with the color profile will be converted to a new color format.

The function:

```
IG_cpm_image_profile_set(       HIGEAR hIGear,        LPBYTE lpRawData,        DWORD
dwRawSize,        AT_BOOL  bConvert);
```

is a special case of a working profile set operation that associates a given color profile with a single image. In this case the image becomes associated not with a global working profile but with a locally given profile.

The function:

```
IG_cpm_profile_get(       AT_MODE nColorSpace,        DWORD nProfileGroup,        LPCHAR
pStatusStr,        UINT nStatusSize,        LPUINT lpnStatusLen,        LPDWORD
lpnProfileSize,        LPBYTE lpProfileData,        DWORD dwProfileDataSize);
```

can be used to get the current value of a color profile associated with the color space given by the first argument in the profile group specified by the second argument. Other arguments return information about the profile along with the profile itself.

To get the color profile information associated with a given image, the function:

```
IG_cpm_image_profile_get(       HIGEAR hIGear,        LPAT_BOOL lpbIsLocal,
LPAT_MODE lpnColorSpace,        LPCHAR lpStatusStr,       UINT nStatusSize,        LPUINT
lpnStatusLen,        LPDWORD lpnProfileSize,        LPBYTE lpProfileData,        DWORD
dwProfileDataSize);
```

can be used. It returns information about the local profile if it is associated with an image, or a global profile otherwise. The second argument returns TRUE if the local profile is associated or FALSE if it is not.

The function:

```
IG_cpm_profiles_reset(AT_BOOL bConvert)
```

is introduced to reset all global profiles to their default values. If bConvert is TRUE, then all images associated with the old previous profiles will be converted to the new ones.

For a detailed description of the color profile functions, please see the Core Component API Function Reference.

1.2.4.8 Annotating Images

ImageGear Annotation (ART) component is not available for Mac OS X platform. All related methods behave as if ImageGear ART component is not initialized.

1.2.4.9 Advanced Image Formats

This section provides information about the following:

- Adobe PDF
  - About the PDF Component
  - About PDF Standards
  - Attaching the PDF Component to Core ImageGear
  - Single- and Multi-Threaded Applications
  - Working with PDF Layers
  - Distributing PDF and PS Fonts with Your Application
- DICOM
  - Loading and Saving DICOM Images
  - Processing 9...16-bit Grayscale Images
  - Displaying Medical Grayscale Images
  - Working with DICOM Non-Image Data
    - Associating DICOM Data with an ImageGear Image
    - Reading Data from Data Elements
    - Writing Data to Data Elements
    - Working With DICOM Data Structures
    - Working with DICOM Data Dictionary
  - Working with Presentation State Objects

1.2.4.9.1 Adobe PDF

This section tells how to use ImageGear PDF component.

- About the PDF Component
- About PDF Standards
- Attaching the PDF Component to Core ImageGear
- Single- and Multi-Threaded Applications
- Working with PDF Layers
- Distributing PDF and PS Fonts with Your Application

1.2.4.9.1.1 About the PDF Component

This section provides information about the following:

- PostScript Language (PS)
- Adobe® Portable Document Format (PDF)
- Content Editing
- Document Fonts
- Word Extraction
- Document Metadata
- Native Printing

PostScript Language (PS)

The PostScript Language (PS) is a simple interpretive programming language with powerful graphics capabilities. Its primary application is to describe the appearance of text, graphical shapes, and sampled images on printed or displayed pages, according to the Adobe® imaging model. A program in this language can communicate a description of a document from a composition system to a printing system or control the appearance of text and graphics on a display. The description is high-level and device-independent.

PostScript format is not supported on MacOS X platform.

Adobe® Portable Document Format (PDF)

The Adobe® Portable Document Format (PDF) is the native file format of the Adobe® Acrobat® family of products. PDF relies on the same imaging model as the PostScript page description language to describe text and graphics in a device-independent and resolution-independent manner. A document can be converted between PDF and the PostScript language; the two representations produce the same output when printed. To improve performance for interactive viewing, PDF defines a more structured format than that used by most PostScript language programs. PDF also includes objects, such as annotations and hypertext links that are not part of the page itself but are useful for interactive viewing and document interchange. However, PDF lacks the general-purpose programming language framework of the PostScript language.

Using the ImageGear PDF API allows you to load, save, edit and process native PDF and PostScript documents. The ImageGear PDF Component can also perform rasterization of PDF and PostScript documents, converting them to bitmaps. The component also provides you with the ability to extract text from loaded PDF and PostScript documents

ImageGear PDF component provides full multi-page reading and writing support for the entire document as well as specified set of pages. You can detect, read, write, append, insert, replace, swap and delete a specified page in the PDF document.

Content Editing

Content editing provides an API for creating, accessing and editing PDF page content objects. With this API you can work with a page's content as with a list of such objects like images, texts, forms. Retain, modify and save their data and properties.

Document Fonts

Document font support includes:

- Listing the fonts available in the host system and finding a system font that matches a PDF font
- Creating a font from the system font and encoding as well as from the specified attributes
- Changing font's information
- Editing and embedding a font in the document
- Both single and multiple bytes fonts are supported

Word Extraction

Text words extraction includes:

- Extracting words from a PDF document or specified page
- Enumerating and sorting the words

- Getting word layouts, styles and characters

Document Metadata

You can get and set PDF document metadata corresponding to a document's Info dictionary.

Native Printing

Native PDF document printing now renders the document content directly to the printer, so it is fast and requires less memory.

PDF support is compatible with Adobe® PDF version 1.7 as defined in the Portable Document Format Reference Manual Version 1.7, distributed by Adobe Systems Incorporated. It provides reading capability up to the PDF version 1.7 and writing of the PDF 1.7 documents.

PostScript support is compatible with Adobe®PostScript® 3.0 language as defined in the PostScript Format Reference Manual, distributed by Adobe Systems Incorporated. It provides reading capability up to the PostScript 3.0 Language Level 3 and writing of the PostScript 3.0 files with the Language Level 1, 2 and 3. This includes writing of the Enhanced PostScript (EPS) files with standard and extended preview as well as w/o preview at all.

1.2.4.9.1.2 PDF Standards

PDF/X and PDF/A standards are defined by the International Organization for Standardization (ISO).

- PDF/X standards apply to graphic content exchange
  The most widely used standards for a print publishing workflow are several PDF/X formats: PDF/X-1a, PDF/X-3, and PDF/X-4.
- PDF/A standards apply to long-term archiving of electronic documents
  The most widely used standards for PDF archiving are PDF/A-1a and PDF/A-1b (for less stringent requirements).

For more information on PDF/X and PDF/A, see the ISO website.

PDF/A

PDF/A is a file format based on PDF. It provides a mechanism for representing electronic documents in a manner that preserves their visual appearance over time, independent of the tools and systems used for creating, storing or rendering the files.

PDF/A-1a

Level A conforming files shall adhere to all of the requirements of the ISO 19005-1:2005 specification.

A file meeting this conformance level is said to be a "conforming PDF/A -1a file."

PDF/A-1b

In recognition of the varying preservation needs of the diverse user communities making use of PDF files, the ISO 19005-1:2005 specification defines a Level B conformance level. Level B conforming files shall adhere to all of the requirements of the spec except those applicable to Level A only.

A file meeting this conformance level is said to be a "conforming PDF/A-1b file."

> The Level B conformance requirements are intended to be those minimally necessary to ensure that the rendered visual appearance of a conforming file is preservable over the long term. However, Level B conforming files might not have sufficiently rich internal information to allow for the preservation of the document's logical structure and content text stream in natural reading order, which is provided by Level A conformance. The requirements for Level A conformance place greater responsibilities on writers of conforming files and those preparing such files, but these requirements allow for a higher level of document preservation service and confidence

PDF/X

PDF/X is a PDF based format for the exchange of object-based data where individual objects may be in either vector or raster data structures. PDF/X defines a data format and its usage to permit the predictable dissemination of a compound entity to one or more locations, as color-managed, CMYK, gray, RGB, and/or spot color data, in a form ready for final print reproduction, by transfer of a single file. This file contains all the content information necessary to process and render the document, as intended by the sender, coded inside a single PDF file. No other parts, neither external files nor internally embedded files, are required or permitted. This exchange requires no prior knowledge of the sending and receiving environments and is sometimes referred to as "blind" exchange. It is platform- and transport-independent.

PDF/X-3:2003

PDF/X-3:2003 conforming files shall adhere to all of the requirements of the ISO 15930-6:2003 specification defining "Prepress digital data exchange using PDF -- Part 6: Complete exchange of printing data suitable for color-managed workflows using PDF 1.4 (PDF/X-3)".

A file meeting this conformance level is said to be a "conforming PDF/X-3 file."

1.2.4.9.1.3 Attaching the PDF Component to Core ImageGear

To use the ImageGear PDF Component, you have attach this component to core ImageGear using the function IG_comm_comp_attach:

```
IG_comm_comp_attach("PDF")
```

To check if the Component is attached successfully, use the function IG_comm_comp_check:

```
IG_comm_comp_check("PDF")
```

This function (method) returns TRUE if the Component is attached.

Initializing the Component

To initialize the ImageGear PDF component, the "PDF" component needs to be attached to the ImageGear component manager. Then, each thread that uses PDF functionality must call IG_PDF_initialize. Please see Single- and Multi-Threaded Applications for additional information.

The following resource content is required by the ImageGear PDF component initialization routine.

Resource\PDF\CIDFont\	PDF CID fonts directory
Resource\PDF\CMap\	PDF font CMaps directory
Resource\PDF\Font\	PDF fonts directory
Resource\PDF\Unicode\	PDF unicode mappings directory

Retrieving Info About the Component

To retrieve information about the attached Component, call the following function:

```
      IG_comm_comp_list(LPUINT *lpnCount, UINT nIndex, LPCHAR lpComp, DWORD dwCompSize,
LPUINT lpnRevMajor, LPUINT lpnRevMinor, LPUINT lpnRevUpdate,
LPCHAR lpBuildDate, UINT nBDSize, LPCHAR lpInfoStr, UINT nISSize)
```

This function (methods) provides you with the full list of info about the component determined by nIndex index from the list of currently attached components whose number are returned trough lpnCount argument.

For more detailed information about these functions usage see Using ImageGear Component Manager section

1.2.4.9.1.4 Single- and Multi-Threaded Applications

In a single-threaded application, the initialization (IG_PDF_initialize()) and termination (IG_PDF_terminate()) functions must be called only once during the life of the application. Attempting to initialize the Adobe PDF Library and DLI more than once in the application may cause errors or unpredictable behavior, and is not supported. You are free to create multiple documents and/or multiple files within the run, but the initialization and termination of the Adobe PDF Library and DLI is limited to one iteration of each.

In a multi-threaded application, you must call IG_PDF_initialize() and IG_PDF_terminate() in the main thread and in each thread that will use the ImageGear PDF Component. The reason is that the first initialization of the Library requires some extra processing that later initializations do not, such as building of font resource lists. By keeping an initialized Library instance open in the parent process, you can improve the initialization time for every child process. Otherwise, if you have a point in the process where no child has the Library open, the next one to initialize will have to do a full startup again.

Thus we strongly recommend, whenever possible, that you initialize the Adobe PDF Library in the main process thread before initialization in any other threads, and terminate the Library in the main process thread after terminating in other threads. This will provide enhanced performance when initializing the Library in a process' child threads.

Access to the same PDF document from multiple threads is not permitted because multiple threads cannot share Adobe PDF Library data types. PDF documents created/opened in the main thread can be only used from the main thread.

1.2.4.9.1.5 Working with PDF Layers

This section explains how to work with PDF Layers, as follows:

- Layer Objects and Visibility
- Layers Diagram
- Visibility Policy
- Working with Containers, Dictionaries and Layers using ImageGear PDF sample

For more information, see Using ImageGear PDF Component and HIG_PDF_DICTIONARY and HIG_PDF_LAYER.

Layer Objects and Visibility

The following objects are related to PDF Layers and responsible for the visibility.

Layers

Each layer has a name and a visibility state for the containers connected to the layer through a Dictionary. The visibility state can have one of two values - ON or OFF.

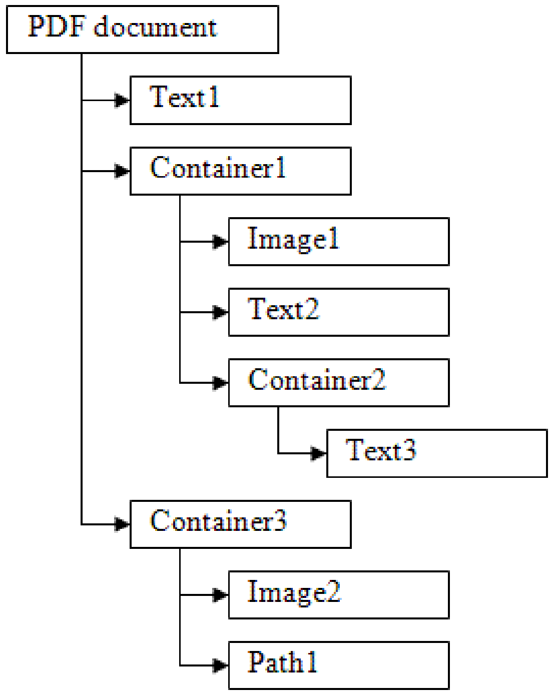
Dictionaries

Each dictionary contains the array of layers and a Boolean function, which takes all the layers' states as input and applies the visibility policy function to the layers' state values producing the Boolean result whether or not to display all the objects from the dictionary's container. The output can have one of two values - ON or OFF.

Containers

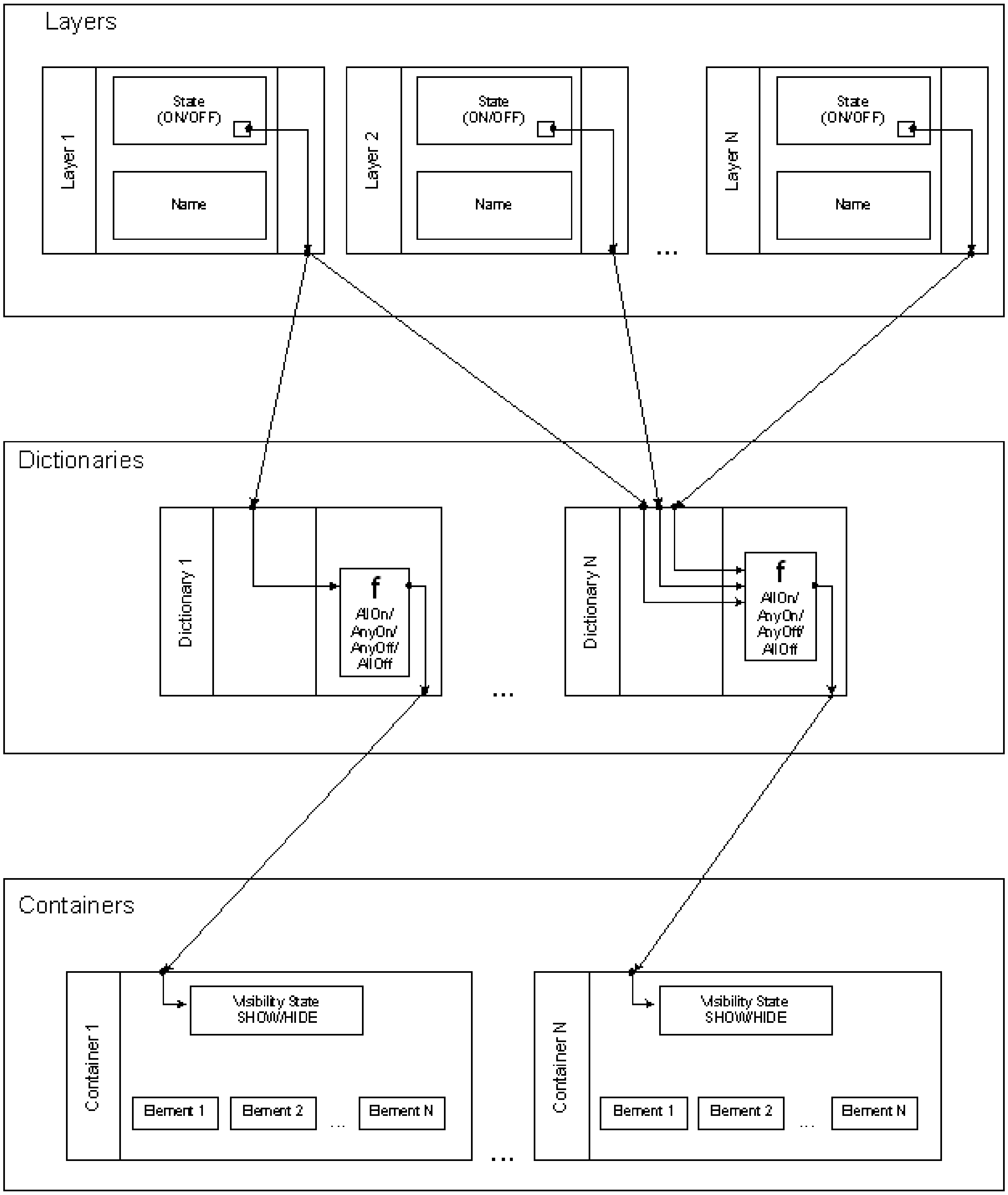
Each container is an arbitrary set of PDF elements or other containers. Each container is associated with a dictionary. The container and all its elements (including the other containers) are displayed when its dictionary's visibility policy Boolean function results with ON.

The following diagram shows the relationship between containers and the other PDF document objects:



Layers Diagram

The following diagram shows interconnection between PDF objects and layers:



Visibility Policy

Dictionary Boolean functions define the visibility policy. It can be one of the following:

- AllOn - equivalent to "AND" function for all input parameters,
- AnyOn - equivalent to "OR" function for all input parameters,
- AnyOff - equivalent to "NOT-OR" function for all input parameters,
- AllOff - equivalent to "NOT-AND" function for all input parameters,

In other words:

AllOn	All the containers from the dictionary will be displayed when all the input layers' states are `ON'
AnyOn	All the containers from the dictionary will be displayed when at least one of the input layers state is `ON'
AnyOff	All the containers from the dictionary will be displayed when at least one of the input layers state is `OFF'
AllOff	All the containers from the dictionary will be displayed when all the input layers' states are `OFF'

Working with Containers, Dictionaries and Layers using ImageGear PDF sample

Before creating a container you need to create one or more layers using the View -> Layers menu. Make sure to create some simple elements like text, paths or images as well.

In order to create container right click on the PDF page and select Create -> Container. You will be prompted to select all the layers you want to associate with the containers dictionary, which will affect the container's visibility. You also need to select the visibility policy and check all the elements you want to include into the new container. When container is created a new dictionary is automatically created as well.

To show or hide the container's elements use View -> Layers menu to change the layer's state to ON or OFF.

In order to rearrange the container's elements you can right click to delete the existing container, which will free its elements and create the new one.

1.2.4.9.1.6 Distributing PDF and PS Fonts with Your Application

The ImageGear Professional toolkit comes with multiple PDF and PS fonts that can be used for developing an application based on ImageGear.

These fonts are the property of Adobe® Systems and are fully licensed for distribution with your application.

If you need to distribute additional fonts, you need to get a license for the redistribution of those fonts. It's possible to add other fonts to be used with ImageGear PDF/PS support. To do this, place your fonts to the PDF "Resource" directory as follows:

- CID fonts (if any) to ...\Resource\PDF\CIDFont
- CMaps fonts to ...\Resource\PDF\CMap
- PDF fonts to ...\Resource\PDF\Font
- PostScript fonts to ...\Resource\PS\Fonts

Please also note that the fonts that are pre-licensed for distribution (i.e., AdobeSansMM, AdobeSerifMM, "Courier" and "NotDefFont") provide the substitution capabilities for the other fonts, so they are likely to be enough for many cases. The below paragraph provides more technical details regarding font substitution.

The multiple master font format is an extension of the Type 1 font format that allows the generation of a wide variety of typeface styles from a single font program. This is accomplished through the presence of various design dimensions in the font. Examples of design dimensions are weight (light to extra-bold) and width (condensed to expanded). Coordinates along these design dimensions (such as the degree of boldness) are specified by numbers. A particular choice of numbers selects an instance of the multiple master font. Adobe® Technical Note #5015, Type 1 Font Format Supplement, describes multiple master fonts in detail.

1.2.4.9.2 DICOM

The MD (Medical) component is a full-featured ImageGear component. It supports the DICOM format, contains a custom API, and includes expanded image processing capabilities beyond those of the baseline ImageGear library.

This section provides information about the following:

1.2.4.9.2.1 Loading and Saving DICOM Images

With the MD component loaded, your application is ready to load and save DICOM image files in addition to the file formats normally supported by ImageGear.

See DICOM in the File Formats chapter for detailed information about ImageGear support for the DICOM format.

Loading DICOM Images

Load DICOM images in the same way as you load images of all other formats, using one of the Formats Component loading methods. You can use Page or Document loading modes, and all Image Sources that are supported by the Formats Component. Use DICOM Filter Control Parameters to control loading of the DICOM images.

Saving DICOM Images

There are several ways to save DICOM images using ImageGear Medical. The functions included as part of the MD component API have a number of DICOM specific parameters. The others are included in the baseline API. You may need to pass the DICOM parameters to these functions via the Filter Control Parameters. You can also use the ImageGear's Multi-page Image API to load and save multi-page (cine) DICOM images. The saving functions are listed below.

Description	**DLL**
DICOM-specific saving function	MED_DCM_save_DICOM()
DICOM-specific saving function for a file that has already been opened and for which you have a File Descriptor handle	MED_DCM_save_DICOM_FD()

One of the baseline ImageGear saving functions	IG_save_file()
	IG_save_FD()
	IG_save_mem()
	IG_fltr_save_file()

The important consideration in writing a DICOM image is whether the image being saved out was originally a DICOM image. If the image was originally a DICOM image (and assuming that you have not altered its Data Set to contain illegal values), the saving of such an image is quite a simple process. This chapter describes how to save an image that has a valid Data Set, how to use the DICOM-specific saving functions, and what Image Control options can be used to save the image when using the baseline functions.

This section provides information about the following:

- Using the DICOM Specific Saving Function
- Saving a DICOM Image Using a File Descriptor
- Saving a DICOM Image Using Baseline ImageGear API Calls
- Saving DICOM Images with JPEG Compression
- Saving out a DICOM Image from a Non-DICOM Image
- Saving a Multi-Frame DICOM Image
- Saving DICOM Images Using ImageGear's Multi-Page Image API

Using the DICOM Specific Saving Function

MED_DCM_save_DICOM() saves the DICOM file using the name specified by the first argument. Here is a sample call:

```
MED_DCM_save_DICOM("OmyBack.dcm", hIGear,
MED_DCM_TS_JPEG_LOSSY_8, TRUE, TRUE,
MED_DCM_PLANAR_PIXEL_BY_PIXEL, TRUE, 100, 0);
```

Notice that the setting for Transfer Syntax in the sample call indicates that JPEG compression should be used on the file. For this reason the second-to-last argument nJPEGQuality is used. This is the quality setting for lossy JPEG compression. The setting shown (100) results in the highest quality that is possible using this compression scheme. If another Transfer Syntax had been specified, the setting of nJPEGQuality would have been meaningless.

Note that DICOM is a multi-page format. When you save an image into existing DICOM file, ImageGear tries to append this image at the end of the file. If for some reason appending of the image is not possible, ImageGear returns an error. If you do not want to append a page, delete the existing file before saving to that filename, or use IG_fltr_save_file() with the bOverwrite parameter set to TRUE.

Saving a DICOM Image Using a File Descriptor

If you are saving a DICOM image to a file for which you have a File Descriptor handle, call MED_DCM_save_DICOM_FD(). Here is a sample call:

```
      MED_DCM_save_DICOM_FD(fd, hIGear, MED_DCM_TS_EXPLICIT_VR_LE, FALSE, TRUE,
MED_DCM_PLANAR_PIXEL_BY_PIXEL, TRUE, 0, 0);
```

Saving a DICOM Image Using Baseline ImageGear API Calls

ImageGear baseline API functions can also be used to save a DICOM image. However as these functions do not contain any of the custom parameters for saving a DICOM image, you may need to make some Filter Control calls first. You will find the list of available DICOM filter control parameters for saving in the section DICOM Filter Control Parameters.

Here is an example of setting DICOM filter control parameters for saving and then calling the baseline saving function. All of the parameters values used below are different from the component's default values:

```
/* get current value of SAVE_SYNTAX control parameter */
IG_fltr_ctrl_get(IG_FORMAT_DCM, "SAVE_SYNTAX", FALSE, NULL, NULL,
(LPVOID)nSaveSyntax, sizeof(nSaveSyntax));
/* set new value to SAVE_SYNTAX control parameter */
IG_fltr_ctrl_set(IG_FORMAT_DCM, "SAVE_SYNTAX", (LPVOID)MED_DCM_TS_EXPLICIT_VR_LE,
sizeof(nSaveSyntax));
/* get current value of SAVE_SMALLEST control parameter */
IG_fltr_ctrl_get(IG_FORMAT_DCM, "SAVE_SMALLEST", FALSE, NULL, NULL,
(LPVOID)bSaveSmallest, sizeof(bSaveSmallest));
/* set new value to SAVE_SMALLEST control parameter */
IG_fltr_ctrl_set(IG_FORMAT_DCM, "SAVE_SMALLEST", (LPVOID)TRUE,
sizeof(bSaveSmallest));nErrcount = IG_save_file(hIGear, "knee.dcm", IG_SAVE_DCM);
```

Saving DICOM Images with JPEG Compression

ImageGear supports the following modes for saving JPEG compressed DICOM images:

- **JPEG Lossy Baseline or Extended**
  This is the default mode for saving DICOM images with JPEG compression in ImageGear. In this mode, depending on the channel depth of the source image, ImageGear saves it as either 8 bits per channel, using "JPEG Baseline (Process 1)" transfer syntax, or as 9..12 bits per channel, using "JPEG Extended (Process 2 & 4)" transfer syntax.

  Use one of these ways to save an image as DICOM JPEG Baseline or Extended:

  - Save the image using a general ImageGear saving function, such as IG_fltr_save_file, and passing IG_FORMAT_DCM | IG_COMPRESSION_JPEG << 16 to the lFormatType parameter. JPEG control parameter SAVE_TYPE should be set to IG_JPG_LOSSY. DICOM control parameter SAVE_SYNTAX is ignored.
  - Save the image as IG_FORMAT_DCM, using a general ImageGear saving function. Set DICOM control parameter SAVE_SYNTAX to either MED_DCM_TS_JPEG_LOSSY or MED_DCM_TS_JPEG_EXTENDED_PR_2_4. JPEG control parameter SAVE_TYPE is ignored.
  - Save the image using a DICOM-specific saving function. Set nSyntax parameter to either MED_DCM_TS_JPEG_LOSSY or MED_DCM_TS_JPEG_EXTENDED_PR_2_4. JPEG control parameter SAVE_TYPE is ignored.
- **JPEG Lossy Baseline Only**
  This mode provides compatibility with viewers that do not support JPEG Extended coding process (12-bit images). In this mode ImageGear saves images as 8 bits per channel, using "JPEG Baseline (Process 1)" transfer syntax. If the image has greater bit depth, ImageGear reduces it to 8 bits per channel for saving.

  Use one of these ways to save an image as DICOM JPEG Baseline:

  - Save the image as IG_FORMAT_DCM, using a general ImageGear saving function. Set DICOM control parameter SAVE_SYNTAX to MED_DCM_TS_JPEG_BASELINE_PR_1_ONLY. JPEG control parameter SAVE_TYPE is ignored.
  - Save the image using a DICOM-specific saving function. Set nSyntax parameter to

MED_DCM_TS_JPEG_BASELINE_PR_1_ONLY. JPEG control parameter SAVE_TYPE is ignored.

For compatibility with earlier versions of ImageGear, MED_DCM_TS_JPEG_BASELINE_PR_1 constant has the same meaning as MED_DCM_TS_JPEG_LOSSY: the image is saved using either Baseline or Extended process. To save with the Baseline process, make sure to use MED_DCM_TS_JPEG_BASELINE_PR_1_ONLY constant.

- **JPEG Lossless**
  In this mode, depending on the channel depth of the source image, ImageGear saves it as 8...16 bits per channel, using "JPEG Lossless, Non-Hierarchical (Process 14)" transfer syntax.

  Use one of these ways to save an image as DICOM JPEG Lossless:

  - Save the image using a general ImageGear saving function, such as IG_fltr_save_file, and passing IG_FORMAT_DCM | IG_COMPRESSION_JPEG << 16 to the lFormatType parameter (DICOM control parameter SAVE_SYNTAX is ignored). JPEG control parameter SAVE_TYPE should be set to IG_JPG_LOSSLESS.
  - Save the image as IG_FORMAT_DCM, using a general ImageGear saving function. Set DICOM control parameter SAVE_SYNTAX to either MED_DCM_TS_JPEG_LOSSLESS, or MED_DCM_TS_JPEG_LOSSLESS_FIRSTORDER. JPEG control parameter SAVE_TYPE is ignored.
  - Save the image using a DICOM-specific saving function. Set nSyntax control parameter to either MED_DCM_TS_JPEG_LOSSLESS or MED_DCM_TS_JPEG_LOSSLESS_FIRSTORDER. JPEG control parameter SAVE_TYPE is ignored.

Saving out a DICOM Image from a Non-DICOM Image

If you have created a Data Set for a loaded non-DICOM image and wish to save it, please remember that it is up to you to add all the Mandatory Data Elements to make it a valid Part 3 DICOM file. When you create the Data Set, a handful of basic Data Elements are added to it. These values were taken from the image. Note that they are not sufficient to satisfy the requirements of Part 3 of the Standard.

Saving a Multi-Frame DICOM Image

ImageGear MD component allows you to save single-frame as well as multi-frame (multi-page) images. Saving a multi-frame DICOM image is pretty straightforward. Use any standard image saving function from ImageGear baseline API. Image saving functions from Medical API do not support saving of multi-frame images.

Use the following steps to create a multi-frame image:

- Create or load first frame into HIGEAR.
- Create the DataSet if it is not present.
- Add the DCM_TAG_NumberOfFrames tag to the DataSet, if it is not present. Value of this tag is irrelevant but its presence is necessary.
- Save the image using any ImageGear baseline saving function.
- * Create or load second frame into HIGEAR.
- Create Data Set if it is not present.
- Add the DCM_TAG_NumberOfFrames tag to the Data Set, if it is not present.
- Save the image, specifying the same file name that you used for the first frame.

* Repeat for all consequent frames.
Several limitations that are imposed by the DICOM standard are listed below.

All frames of the image should have same dimensions, bit depth and photometric interpretation. All frames should use the same Transfer Syntax (compression). ImageGear returns an error if these conditions are not met.

It is important to note that only a few DICOM modalities allow the presence of multiple frames in the image, while the others do not. When you try to add a frame to an existing DICOM file, ImageGear checks the file for presence of the NumberOfFrames (0028, 0008) Data Element. If it is present ImageGear tries to append the frame, otherwise ImageGear considers the file as belonging to a single-frame modality and returns an error.

Another important fact about DICOM format is that it has a continuous structure which means all the Data Elements as well as image frames go one after another. This results in the following limitations:

- Inserting a frame in a file that contains multiple frames, would lead to rewriting all the consequent frames. The same thing would happen when overwriting a frame in a file that uses some compression. Changed size of the compressed frame would lead to rewriting the whole file. To prevent such unpredictable time-consuming operations, ImageGear Medical only supports appending a frame at the end of the DICOM image, but not insertion or overwriting of frames.
- ImageGear Medical writes the DICOM DataSet only when writing the first frame, and does not modify it when adding frames. The only exclusion is the "Number Of Frames" Data Element, which is always kept consistent with the actual number of frames.

- The "Number of Frames" Data Element has a Value Representation of "Integer String", i.e., it is stored in the file as a character string rather than an integer. This string can occupy up to 14 characters. Now imagine that you are appending frame to a file that has 99 frames. The number 99 occupies two bytes in the file (because it is composed of two digits). After a frame is added, the Number Of Frames DE becomes 100, needing 4 positions for its storage (three digits plus padding). This results in the need of shifting the whole file by two bytes. To prevent this, ImageGear Medical always leaves the maximum of 14 positions for storing the Number Of Frames DE. Moreover, it does not append a frame to a file where Value Length of the Number of Frames Data Element is less than 14 (for example, if the file has been created by some other vendor1).

If you need to insert or delete a frame, or modify DataSet in the DICOM image, create a new image with the modified DataSet, copy necessary frames to it, and then delete the original image.

Saving DICOM Images Using ImageGear's Multi-Page Image API

You can use ImageGear's Multi-page Image API calls (IG_mpi_... and IG_mpf_...) for loading and saving multi-frame DICOM images. Just note that ImageGear Medical only supports the appending of frames but not insertion, deletion or overwriting of frames.

1.2.4.9.2.2 Processing 9...16-bit Grayscale Images

Most of the baseline ImageGear processing functions support 9..16 bit images.

Medical Component presents a set of methods that were designed specifically for 9..16 bit and Medical images.

The Medical Image Processing API is described below:

MED_IP_high_bit_transform()	Gives the high bit data element a new value and transforms the image accordingly.
MED_IP_histo_clear()	Clears the histogram created by MED_IP_histo_tabulate()
MED_IP_histo_tabulate()	Tabulates a histogram for a specified region of an image (or the whole image), and allows you to set the width of and number of bins that are used. This function also indicates whether the image is signed or not.
MED_IP_min_max()	Gets the minimum and maximum pixel values from a specified region of an image (or the whole image), and also indicates whether the image is signed or not.
MED_IP_normalize()	Converts a signed image to an unsigned image. Takes an argument for a minimum pixel value, converts the lowest pixel value in the image to the new minimum, and maps all the rest of the pixels in the image such that the original contrast is maintained.
MED_IP_promote_to_16_gray()	Promotes an 8g or 8i image to a 16-bit grayscale image and gives you the possibility to select the position of the high bit.
MED_IP_reduce_depth_with_downshift()	Reduces a 9-16-bit image to an 8-bit image using downshifting, that allows you to choose which 8 bits to use for the new image.
MED_IP_reduce_depth_with_LUT()	Reduces a 9-16-bit image to an 8-bit image using your own LUT or the current display16x8 LUT.
MED_IP_swap_bytes()	Corrects a poorly constructed image in which the bytes of each 16-bit pixel are in the wrong order.
MED_IP_contrast()	Reduces a 9-16-bit image to an 8-bit image using Rescale Slope, Intercept, Window Center, Width and Gamma. You would normally use Rescale and Window settings from the Data Set, and your own setting for Gamma.
MED_IP_contrast_auto()	Same as MED_IP_contrast() except that it automatically derives Window Level and Window Center values for you by scanning the image for the min/max pixel values.

Critical Data Elements

In this manual, we use the term "Critical Data Elements" for those DEs that are used to help load a DICOM image correctly. It's important to note that when certain IP methods are called, the toolkit alters the Data Set. See the "Critical Data Elements" section of Working With DICOM Data Structures for more details.

1.2.4.9.2.3 Displaying Medical Grayscale Images

Many medical images use more than 8 bits per grayscale pixel. The main concern in displaying such images is that commonly used monitors can display only 256 shades of gray, which corresponds to 8 bits per pixel. The 16-bit pixels need to be mapped in some way to 8-bit pixels. ImageGear uses two approaches for this mapping.

If a 9..16 bit grayscale image has been loaded from a non-DICOM file, its pixels are mapped to 8g by left shifting the pixels by (n-8), where n is the bit depth of the image. For example, if the image has 12 bits per pixel, its pixels will be left shifted by 4 bits, so the 8 most significant bits of the 12-bit pixel will be used. This mapping is done only for image display, and does not affect the image stored in memory.

When ImageGear loads a DICOM image, it creates a 16-bit to 8-bit, or 8-bit to 8-bit display Look-Up Table (16x8 LUT or 8x8 LUT), and attaches it to the image. This LUT gives more flexibility in displaying medical images, allowing to display a certain range of pixel intensities with best contrast.

DICOM image files may contain several Look-Up Tables that describe how the image shall be displayed. "Modality" LUT specifies transform of image pixels into modality meaningful values, such as optical density or Hounsfield Units. "Value of Interest" LUT (VOI LUT) specifies what range of pixel intensities should be shown on the screen. Most often, both these LUTs are linear, and thus are presented by a pair of values that are similar to brightness and contrast. For Modality LUT, these are Rescale Intercept (0028, 1053) and Rescale Slope (0028, 1053). For VOI LUT, these are Window Center (0028, 1050) and Window Width (0028, 1051). The standard also allows the usage of non-linear LUTs. These LUTs are represented as an array of values that map source image intensities to the output range.

If all of these values are found in the file, they are all used to build the 16x8 LUT. If Rescale values are not found, default values (Intercept = 0.0, Slope=1.0) are substituted. If VOI LUT values are not found, the image is scanned for min and max pixel intensities, and the LUT is built to display the min intensity as black and max intensity as white. The values between min and max are linearly scaled between black and white.

You can adjust the values in the 16x8 LUT using MED_display_...() functions.

Grayscale LUTs can be attached to a HIGEAR or to a Display Group. In the latter case, if you are using multiple Display Groups corresponding to the same HIGEAR, they can have different LUTs, allowing to display the same image with different contrast settings simultaneously, in different windows. See the "Medical Component Grayscale Look-Up Tables" section below for more details.

Grayscale LUTs (16x8, 8x8) only work with grayscale images. They do not work with bi-tonal, indexed (paletted) or color images.

Pixel Padding Value

DICOM images sometimes contain a Data Element called "Pixel Padding Value" (PPV). The PPV is used mostly to fill in the corners of round images. DICOM provides a Tag for PPV which is (0028,0120). This Data Element stores a 16-bit grayscale value that is to be treated as the Pixel Padding Value. Any pixels in the image that have this value are not to be treated as meaningful objects - but as background color.

When the ImageGear Medical loads a DICOM image that contains a PPV the value is captured and stored in the HDS, which is attached to the new image. In fact, 3 values are stored to the HDS: the PPV from the PPV Data Element, a flag indicating that a PPV was found in the file when it was loaded, and an 8-bit grayscale value used to display pixels with this value.

Use MED_DCM_DS_PixPadVal_get() and MED_DCM_DS_PixPadVal_set() functions to get/set the Pixel Padding Value that is to be used while displaying a 16-bit grayscale image.

Pseudocoloring Medical Images

Medical Display API contains methods for creating color LUTs that can be used to pseudocolor grayscale images.

MED_display_color_create()	Chooses a pre-defined ImageGear pseudo color scheme and creates 3 LUTs for the RGB components that are ready for use with display. You can call MED_display_color_set() to associate these RGB LUTs with an image.
MED_display_color_limits()	Displays over and under-saturated areas with pseudo color of your choice.
MED_display_color_set()	Associates 3 LUTs with an image. Can be used to apply pseudo color to an image. You can set to your own LUTs, to the LUTs created by MED_display_color_create(), or set to NULL to use linear 0-255 grayscale LUTs.

Medical Component Grayscale Look-Up Tables

The ImageGear Medical component provides a set of functions that allow you to create grayscale look-up tables according to DICOM display attributes, such as VOI LUT, Modality LUT, Presentation LUT, etc.

Use the IG_LUT_create() function to create a grayscale LUT, then use MED_display_grayscale_LUT_build() function to fill this LUT with values corresponding to DICOM display settings. This function supports both linear and non-linear Modality and VOI LUTs, as well as presentation state related LUTs.

Once you have built the LUT, you can copy it to either the image, or to the image display settings, by using IG_image_grayscale_LUT_update_from() or IG_dspl_grayscale_LUT_update_from().

IG_image_grayscale_LUT_... API internally uses the same LUT as the one that can be set by IG_display_option_get/set functions.

When ImageGear loads a grayscale DICOM image, it builds a grayscale LUT and attaches it to the image. This LUT can be obtained with the IG_image_grayscale_LUT_copy_get() function.

Example:

```
AT_MED_DCM_DISPLAY_SETTINGS DICOMDisplaySettings;
HIGLUT GrayLUT = (HIGLUT)NULL;
memset(&DICOMDisplaySettings, 0, sizeof(DICOMDisplaySettings));
if (MED_DCM_DS_LUT_exists(g_hIGear, g_hIGearPresState,
DCM_TAG_ModalityLUTSequence))
{
        MED_DCM_DS_LUT_copy_get(g_hIGear, g_hIGearPresState,
DCM_TAG_ModalityLUTSequence, &DICOMDisplaySettings.ModalityLUT);
}
else
{
        DICOMDisplaySettings.ModalityRescale.Slope = 1.0;
        DICOMDisplaySettings.ModalityRescale.Intercept = 0.0;
}
DICOMDisplaySettings.VOIWindow.Center = 1024;
DICOMDisplaySettings.VOIWindow.Width = 2048;
DICOMDisplaySettings.Gamma = 1.0;
IG_LUT_create(12,               /* Input depth of images to use this LUT for. */
        FALSE,          /* Apply to unsigned images. */
        8,              /* Display bit depth to use this LUT for. 8-bits is common for
most PC
monitors. */
        FALSE,          /* Apply to unsigned displays. */
        &GrayLUT);
/* Build a grayscale LUT based on display settings */
MED_display_grayscale_LUT_build(&DICOMDisplaySettings, GrayLUT);
/* Copy LUT to the image. */
IG_image_grayscale_LUT_update_from(g_hIGear, GrayLUT);
IG_LUT_destroy(GrayLUT);
```

See Also:

Working with Grayscale Look-Up Tables

Grayscale Look-Up Tables

1.2.4.9.2.4 Working with DICOM Non-Image Data

This section provides information about the following:

- Associating DICOM Data with an ImageGear Image
- Reading Data from Data Elements
- Writing Data to Data Elements
- Working With DICOM Data Structures
- Working with DICOM Data Dictionary

1.2.4.9.2.4.1 Associating DICOM Data with an ImageGear Image

As a DICOM file is being loaded, the ImageGear Medical associates some additional structures with it. DICOM Data Elements are stored into two separate chunks: a table of Data Elements, which holds those Data Elements that make up the core of the image file, and a Part 10 Template, which holds the Data Elements from the Part 10 Header. The Part 10 Header is kept separate due to some of its special characteristics.

The table of Data Elements is internal and is only accessible via the API functions provided by this component. You can remove it (and free its memory), or create a new table if it does not exist. The names of the API functions that allow you to access the data in the internal Data Set table all start with MED_DCM_DS_ .

We refer to the Part 10 Header area as a "template" because it should be created for every loaded DICOM image, even if the image's Data Set is empty. This template always contains the same set of storage fields for Data Elements-even if the Part 10 Header of the loaded image has no value for them. Further on in this section you will find information on how to get and set Part 10 Header values.

The Part 10 Template cannot be removed. However, as you are saving the image you can choose not to save it as "Part 10-compliant" and the image will be saved without this header.

1.2.4.9.2.4.2 Reading Data from Data Elements

The functions that return you the data for a DE's Data Field include:

MED_DCM_DS_curr_data_get()	Returns the data and size of the data
MED_DCM_DS_curr_data_get_string()	Returns the data as a string and returns the size of the string.

Both of these functions return you the contents of the Data Field of the Current Data Element.

To interpret the returned data call MED_DCM_DS_curr_info_get(). This function returns, among other things, the VR, the VL, and the Item Count.

If the data is binary, i.e. defined as an INT, WORD, etc., you can use the VL and Item Count to determine the length of each item in the Data Field. Binary items aren't delimited; they are stored end-to-end. Since all binary items must be of the same length, you can calculate the length of each item this way:

lengthOfEachItem = VL/Item_count

Using Item_count as a loop delimiter, you can now parse through the data by jumping lengthOfEachItem bytes for as many items as it's returned by Item_count:

```
for (item = 0; item < Item_count; item++)
    {
    int_array[item] = ((int *)value_field)[item];
    }
```

Character data items can be of either fixed length or variable length. In either case, you can forgo the calculation shown above for binary data because character data must be delimited by backslashes.

1.2.4.9.2.4.3 Writing Data to Data Elements

The following API functions allow you to write or overwrite the data of a Data field, respectively:

MED_DCM_DS_DE_insert()	Inserts a new DE to the Data Set. Its placement are determined automatically according to its Tag value. Your only power of placement is in specifying what level in the hierarchy of the Data Set to place the new DE.
MED_DCM_DS_curr_data_set()	Overwrites the Data Field of an existing Data Element.

Whether you are inserting a new DE or overwriting the Data Field of an existing DE, you may need to query the VR and VM by calling MED_DCM_util_tag_info_get(). With the VR in hand, you can call MED_DCM_util_VR_info_mode()which tells you the length that each item can have, and whether the items are of fixed or variable length. See the previous section "Reading Data From Data Elements" for how to interpret the value returned to you in VM.

This section provides information about the following:

- The Components of the Data Set
- The Internal Data Set vs. the Original Data Set
- Critical Data Elements
- The Hierarchy of the Data Set
- Data Set Levels
- Part 10 Header Access

The Components of the Data Set

A Data Set is composed entirely of Data Elements. In DICOM, all stored data, including the images themselves, is stored in fundamental DICOM building blocks calledData Elements.

Access to the Data Set is achieved by moving an internal Data Element index called the "Current Data Element" from Data Element to Data Element. There is a set of API functions, which just move the Current Data Element about the Data Set. Another group of API functions will allow you to retrieve or set the Value Field of the Current Data Element.

The Internal Data Set vs. the Original Data Set

The internal Data Set is similar to but not exactly like the Data Set found in the DICOM file. The DICOM standard has many options and various methods of storing its data. As the internal Data Set is being filled, the vital information is stored in an abstract form. The new form of this information makes it easier for you to work with. Most of this internal storage is completely transparent for you and your application, but there are a few items you should be aware of.

First of all, as you move to see the internal Data Set and inspect its contents you may notice that some items that were in the original Data Set are missing. One example of this are the "Group Lengths" (all of which have Element Numbers of 0000). Group Lengths which are optional in the DICOM specification, are placed in the Data Set to aid to quickly find certain Data Elements by allowing the parser to skip over large blocks of Data Elements that do not contain the Data Element being searched for. They serve no other purpose and are therefore removed. If you are going to save the Data Set back out to disk you may request that Group Lengths should be included in the new disk file. If you do so, ImageGear recalculates the Group Lengths in case any Data Elements have been added, removed or altered.

Also missing from the data is the Data Field for Pixel Data (7FE0,0010). It is the Data Element that holds the image. This Data Element is actually present but his Data Field is empty. This is because the image has been read and loaded into an ImageGear DIB.

Another difference from the internal Data Set and the original file version is that regardless of what the original encoding scheme was, the internal Data Set always are "Explicit VR." That is, the Value Representations (VRs) of each Tag are looked up in the Data Dictionary and recorded along with the other information. If you later write the Data Set to disk you may choose whether the Data Set should be Explicit or Implicit VR.

Critical Data Elements

"Critical Data Elements" are those DEs whose values are taken from the image not regarding to whether the Data Set has values for these DEs or not. Two examples of such DEs are the height (Rows) and width (Columns) of the image. Note that the Critical Data Elements in the internal Data Set are always kept consistent with the HIGEAR. For example, if you have resized the image, the DCM_TAG_Columns and DCM_TAG_Rows Data Elements are set to the new Width and Height of the image. When an image processing function, such as Resize or Rotate, is applied to the DICOM image, the values for Rows and Columns will be updated from the HIGEAR.

The Hierarchy of the Data Set

A certain type of Data Element can be hierarchical meaning that it can have other groups of Data Elements "under" it similarly to the way files are stored under the directory structure. This special type of Data Element has a Value Representation of Sequence Delimiter (SQ). This Data Element l marks a set of Data Elements called "Items." This set can contain 0 or more embedded Data Elements. The hierarchical structure can make it difficult to keep track of where you are as you move through the Data Set. This is the reason of using the Current Data Element - to keep track of index you are positioned at in the array of Data Elements.

Data Set Levels

You may notice that many of the Data Set Navigation API contain a parameter of level_op. This argument tells whether the Data Set index should remain in the current level or can be moved to another level. The possible values for this argument are:

- MED_DCM_MOVE_LEVEL_FIXED
- MED_DCM_MOVE_LEVEL_FLOAT

Part 10 Header Access

When a DICOM image file is loaded it may or may not begin with a Part 10 Header. The Part 10 Header consists of 2 parts: a free-form block of 128 bytes called a Preamble, and a fixed list of 9 Data Elements, all of which have a Group Number of 0002. Both of these parts are stored in the Part 10 Template mentioned earlier. The use of the Preamble is up to the application and can be used to store anything you like, as long as it is 128 bytes or less. A blank or empty Preamble is indicated by 128 bytes of 0x00.

If the original image has a Part 10 header, the data from it is extracted and placed in the internal template mentioned earlier. If there is no header, the template is empty except for 2 fields: the File Meta Information Version (0002,0001) and the Transfer Syntax UID (0002,0010). Default values are assigned to these fields - File Meta Information Version is set to 0x0001, and Transfer Syntax UID is set to the Transfer Syntax detected by ImageGear when the image was loaded. The Version number is called for the DICOM specification and it value never need to be altered.

Access to the values in the Part 10 Template is done using a small set of methods or functions, which include either "Part10" or "preamble" in their names.

MED_DCM_DS_preamble_get()	get Part 10 Header Preamble
MED_DCM_DS_preamble_set()	set Part 10 Header Preamble
MED_DCM_DS_part10_get()	get Part 10 Header Data Elements, except Preamble
MED_DCM_DS_part10_set()	set Part 10 Header Data Elements, except Preamble

1.2.4.9.2.4.4 Working With DICOM Data Structures

When a DICOM image is loaded into ImageGear, it is loaded into a DIB just like any other supported image. In addition to the actual image, the following DICOM-specific data structures are also loaded: File Meta Information Header (if present) and the Data Set. This section explains how to read and/or manipulate the data in the Data Set or File Meta Information Header, or how to create your own Data Set or File Meta Information Header.

This section provides information about the following:

- Getting and Setting Data Set and Part10 Header Data
- Inserting a New Data Element
- Deleting a Data Element
- Getting and Setting Data of the Current Data Element
- Creating a Data Set
- Deleting a Data Set
- Creating a Part 10 Header
- Updating DICOM DataSet without Updating Pixel Data
- Critical Data Elements

Getting and Setting Data Set and Part10 Header Data

With the exception of the first function listed below, these functions can be used for getting or setting various parts of the Part 10 Header data. These functions are useful when you are creating a new DICOM file.

The first function listed below simply returns some information about the Data Set attached to the HIGEAR. This returns the number of Data Elements in the Data Set.

The second function warrants a bit of explanation. When a DICOM image is loaded into ImageGear, the original Transfer Syntax is stored in the HIGEAR. This value is called the Original Transfer Syntax and cannot be changed. You can to read this information using MED_DCM_DS_orig_TS_get(). Even though you can alter the Transfer Syntax Data Element in the Part 10 Header template, the Transfer Syntax of the original image file is kept in case you ever need to know what it was. It does not affect the saving of a DICOM file. To set the Transfer Syntax for an image that you are saving use either the Image Control setting or fill in the TS parameter of MED_DCM_save_DICOM().

MED_DCM_DS_info_get()	Returns the number of DEs in the Data Set
MED_DCM_DS_orig_TS_get()	Returns the Transfer Syntax used to create the image. It is returned as a MED_DCM_TS_ constant.
MED_DCM_DS_part10_get()	Returns the data from a Part 10 item. You specify the item you would like to read. You must supply the item as DCM_PART10_ITEM_ constant.
MED_DCM_DS_part10_set()	Sets the data of an item from the Part 10 meta-info header. You specify which item and supply it with new data to be stored to that item. You must supply a DCM_PART10_ITEM_ constant.
MED_DCM_DS_preamble_get()	Gets the Preamble from the Part 10 header. Your receiving buffer must be at least 128 bytes. You could also use MED_DCM_DS_part10_set() with the appropriate constant to get this value.
MED_DCM_DS_preamble_set()	Sets the Preamble of the Part 10 header. You must supply it with the address of the Preamble and the length of the data you are saving to it. You could also use MED_DCM_DS_part10_get() with the appropriate constant to get this value.

Inserting a New Data Element

ImageGear Medical allows you to insert any number of Data Elements to an existing Data Set, or to a new Data Set that you have created by calling MED_DCM_DS_create(). (See the section "Creating a Data Set" below.)

DICOM Data Sets are sorted numerically by Tag number. Therefore, the Current Data Element does not affect the index position where your new Data Element is inserted. ImageGear takes your Tag number and automatically sorts the new DE into the correct position in the array. The Current Data Element does, however, determine the level at which the new DE is stored. For example, if the Current Data Element is in level 0 and has a Tag value of (0028, 0010) and your new DE has a Tag value of (0028, 0015), the new DE will be inserted into level 0 and positioned somewhere below the Current DE so that its unique Element Number fits numerically in ascending order.

If a DE with specified Tag number already exists in the Data Set, it will be overwritten.

If there are no DEs in the Data Set, the level for the new DE will be 0 or the "top level".

You have complete flexibility in what kind of DEs you can add to the Data Set. For each new DE, you have to supply ImageGear with a Tag structure, a VR, and the data. You can add it to any Group, to any position, with any VR, and with any data that you desire.

An example of using MED_DCM_DS_DE_insert():

```
HIGEAR hIGear;
AT_DCM_TAG tag = 0x00111111;
AT_DCM_VR vr = MED_DCM_VR_CS;
char data[256] = "Data for test element";
DWORD dwSizeOfData = strlen(data);
MED_DCM_DS_DE_insert(hIGear, tag, vr, data, dwSizeOfData);
```

This function requires you to supply the following values: a Tag, a VR, a Data Field, and the size of the data for the Data Field. Below are brief descriptions of what you would need to know in order to set these arguments.

This section provides information about the following:

- Setting the Tag Value
- Setting the VR
- Setting the Data for a VR of CS
- Setting the Data Field
- Setting the Length of a Data Field
- Inserting a Sequence of Data Elements

Setting the Tag Value

All standard Tag values for a DICOM file are assigned constants in enumIGMedTag enumeration. You can also add your own user-defined Tag using MED_DCM_util_tag_info_add().

Setting the VR

The DICOM specification assigns specific VRs for each defined public Tag. In other words, for a particular Tag, you must always use the specified VR. There are a few rare exceptions to this rule, and it is for this reason that you should tell MED_DCM_DS_DE_insert() what VR the data has.

If you don't know the VR for a particular Tag that you are adding, you can find out easily by calling the function MED_DCM_util_tag_info_get(), which returns you its VR as a constant.

Setting the Data for a VR of CS

There are no constants in ImageGear defined for those Data Elements which have a VR of CS. Refer to Part 3 of the DICOM specification for the valid Code Strings which you can enter for data of type CS.

Here is a sample call to insert a Data Element with a Tag of DCM_TAG_PhotometricInterpretation, which has a VR of CS (Code String). You must use one of the defined values from Part 3 of the DICOM specification. The allowed values are: "MONOCHROME1", "MONOCHROME2", "PALETTE COLOR", "RGB", "HSV", "ARGB", "CMYK", "YBR_FULL", "YBR_FULL_422", and "YBR_PARTIAL_422".

```
MED_DCM_DS_DE_insert(hIGear, DCM_TAG_PhotometricInterpretation, MED_DCM_VR_CS,
"MONOCHROME2", 11);
```

Setting the Data Field

What kind of data can be stored to the Data Field depends on the Tag type, the VR, the restriction flags and the Value Multiplicity (VM) defined for the Tag. It is very helpful to call MED_DCM_util_VR_info_mode()or MED_DCM_util_VR_info_string()so that you can find out what kind of restrictions are placed on the kind of data that you would like to enter. Some types of Data Fields are tricky to work with. VRs of Person Name and Code String place a number of restrictions on what can go into the Data Field, and how it is to be formatted. For these VR types, you must consult Part 5 of the DICOM specification.

The lpData parameter is a binary buffer, which can contain a character string, a BYTE or WORD array, or one or several of numbers, such as Integers or Floats. Below there is an example how to set a value of VR=FD (double):

```
DOUBLE dblRefPixelValueX = 1.234;
MED_DCM_DS_DE_insert(hIGear, DCM_TAG_ReferencePixelPhysicalValueX, MED_DCM_VR_FD,
&dblRefPixelValueX, sizeof(DOUBLE));
```

Setting the Length of a Data Field

Because some VRs allow a range of values, the length of all Data Fields cannot be assumed. For this reason, you must supply ImageGear with the length of your data.

Inserting a Sequence of Data Elements

Use the following steps to create and fill a sequence (SQ) of Data Elements:

1. Insert the Data Element of VR = SQ. This positions you one level lower in the DataSet hierarchy. At this level you can only add items to the sequence, not the Data Elements.
2. Insert the DCM_TAG_ItemItem Data Element to denote the beginning of a new item in the sequence. This positions you one level lower in the DataSet hierarchy. Now you can add Data Elements to the embedded Data Set.
3. Insert the data elements you would like to insert to this item.
4. Insert the DCM_TAG_ItemDeliminationItem Data Element to denote end of the item. This positions you one level higher. Repeat steps 2-4 for every item you would like to add.
5. Insert the DCM_TAG_SequenceDeliminationItem Data Element to denote the end of the sequence. This moves you one level higher - to the same level where you added the SQ Data Element.

Examples on creating a Data Element Sequence are below. Note that it is completely valid to add same Data Elements to different Items within a Sequence.

```
AT_DCM_TAG  tag;
char        data_str[64];
tag = DCM_TAG_ReferencedStudySequence;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_SQ, NULL, 0);
tag = DCM_TAG_ItemItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);
tag = DCM_TAG_ReferencedSOPClassUID;
strcpy( data_str, "1.23.456.7.8.90.1234567890.2");
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_UI, data_str, strlen(data_str));
tag = DCM_TAG_ReferencedSOPInstanceUID;
strcpy( data_str, "1.23.456.7.8.90.1234567890.2.1");
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_UI, data_str, strlen(data_str));
tag = DCM_TAG_ItemDelimitationItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);
tag = DCM_TAG_ItemItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);
tag = DCM_TAG_ReferencedSOPClassUID;
strcpy( data_str, "1.23.456.7.8.90.1234567890.3");
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_UI, data_str, strlen(data_str));
tag = DCM_TAG_ReferencedSOPInstanceUID;
strcpy( data_str, "1.23.456.7.8.90.1234567890.3.1");
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_UI, data_str, strlen(data_str));
tag = DCM_TAG_ItemDelimitationItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);
tag = DCM_TAG_SequenceDelimitationItem;
MED_DCM_DS_DE_insert( hIGear, tag, MED_DCM_VR_xx, NULL, 0);
```

Deleting a Data Element

Data Elements can be deleted by calling MED_DCM_DS_curr_remove(). This function removes the DE designated by the Current Data Element. Therefore, you can use one of the MED_DCM_DS_move_...() functions to position the Current Data Element to the DE you want to remove.

If you've used MED_DCM_DS_move_find() or MED_DCM_DS_move_find_first() to position the Current Data Element, be sure to check the value of the Boolean argument to see whether you successfully found the Tag or Group Number that you were looking for. Otherwise, you might unintentionally delete the wrong DE.

Some calls to MED_DCM_DS_curr_remove() result in deletion of more than one Data Element:

- If you are going to delete a DE with a VR of SQ, all the items and consequently DEs below it will be deleted also.
- If you are going to delete a DE of type Item (FFFE, EOOO), all DEs below it will also be deleted.

The following types of DEs may not be deleted:

- Sequence Delimitation Item (SQD) - (FFFE, EODD)
- Item Delimitation Item (ID) - (FFFE, EOOD)

Trying to delete DEs of these types will have no affect on the Data Set. These type of DEs will be automatically deleted by ImageGear when their accompanying set of DEs are deleted.

Getting and Setting Data of the Current Data Element

Notice that all functions in this group include "curr" as part of their names. Below is a description of these functions..

MED_DCM_DS_curr_data_get()	Returns the data and size of data from the CDE. The data is returned in its native format. Use MED_DCM_DS_curr_info_get() to get the VR in order to know how to handle the data.
MED_DCM_DS_curr_data_get_string()	Same as MED_DCM_DS_curr_data_get() except that the data is always returned as a string.
MED_DCM_DS_curr_data_set()	Sets/overwrites the Value Field (data) of the CDE. Check the VR required. You cannot change the VR of an existing DE.
MED_DCM_DS_curr_index_get()	Returns the index of the CDE. Lets you know where you are in the Data Set.
MED_DCM_DS_curr_info_get()	Returns the following pieces of information about the CDE: Tag, VR, VL, Item Count. This is a very important method. Use it as a precautionary call before modifying info in the CDE or inserting/deleting.

Creating a Data Set

If you have a HIGEAR image to which you would like to attach a Data Set, call MED_DCM_DS_create(). This file might have been loaded from a non-DICOM format or may be a DICOM file in which you've destroyed the Data Set (see the section above).

This function creates an empty Data Set and attaches it to the ImageGear's image. The function creates a Data Set structure (outlined above) and fills it with a few Data Elements whose values can be derived from the image. This function also requires you to supply a Transfer Syntax that is saved in the Part 10 Header template as the original Transfer Syntax (as if it was loaded from a DICOM file). If you then save the image and specify a different Transfer Syntax, you overwrite the TS value specified at the time of Data Set creation.

Deleting a Data Set

If you have loaded a DICOM image file and do not need to keep the Data Set around you can free up the memory used to store all the Data Elements by calling MED_DCM_DS_destroy(). Once this function is called the remaining image is no different than any other ImageGear image that was created from any other non-DICOM format. The removed Data Set is discarded and cannot be recovered.

Creating a Part 10 Header

When a DICOM image is loaded with the ImageGear Medical, a Part 10 Header Template is initialized automatically. If you save the image as a Part 10-compliant image (which is the default option), the values from this template are used to fill the header saved with the file.

For those wishing to modify or read the values stored in the Part 10 Header Template, the component supplies 11 constants. All except the last one correlate directly to the fields found in a Part 10 Header. The last constant can be used by ImageGear to store the length of the Private Info field, and is for informational purposes.

Below is the list of Part 10 constants. Each one is followed by its size in bytes. A check appears next to each field that is considered "Mandatory" by the specification. ("Mandatory" is one of three possible "Types" for a DICOM DE. The other Types are "Optional", and "Mandatory Depending on some Condition".)

Mandatory	MD Component Constant	Size of Data Field
Yes	DCM_PART10_ITEM_PREAMBLE	128 bytes
Yes	DCM_PART10_ITEM_VERSION	2 bytes

Yes	DCM_PART10_ITEM_MSSOPCLASSUID	Max 64 bytes
Yes	DCM_PART10_ITEM_MSSOPINSTUID	Max 64 bytes
Yes	DCM_PART10_ITEM_TRANSSYNTAXUID	Max 64 bytes
Yes	DCM_PART10_ITEM_IMPLCLASSUID	Max 64 bytes
	DCM_PART10_ITEM_IMPLVERNAME	Max 16 bytes
	DCM_PART10_ITEM_SRCAETILE	Max 16 bytes
	DCM_PART10_ITEM_PRIVINFOCRUID	Max 64 bytes
	DCM_PART10_ITEM_PRIVINFO	any length
	DCM_PART10_ITEM_PRIVINFO_SIZE	DWORD - length of Priv Info - read only

Please note the following about the above constants:

- You can set DCM_PART10_ITEM_PREAMBLE to anything you want as long as it doesn't exceed 128 bytes
- DCM_PART10_ITEM_VERSION is set by the toolkit to a default value of 1. You should not change this.
- DCM_PART10_ITEM_TRANSSYNTAXUID - we set the value of this DE when the image is loaded. You can read this value; you can even set this value. But when the image is saved your setting are ignored.

If you enter data in DCM_PART10_ITEM_PRIVINFO, it is stored with the new Part 10 Header. Also, ImageGear calculates its size and saves this information internally. This read-only size information is not made part of the template but is kept in case you need to know how much memory to allocate when getting the information from DCM_PART10_ITEM_PRIVINFO. DCM_PART10_ITEM_PRIVINFO_SIZE isn't stored to the new Part 10 header.

Here is an example of a Part 10 field being populated:

```
char szPreamble[] = "This DICOM File is MINE!";
DWORD size_of_data;
size_of_data = sizeof(&szPreamble[0]);
MED_DCM_DS_part10_set(hIGear, DCM_PART10_ITEM_PREAMBLE, &szPreamble[0],
&size_of_data);
```

Updating DICOM DataSet without Updating Pixel Data

There are cases when an application needs to update DataSet of a DICOM file, while leaving PixelData unchanged. For example, if the image is compressed using a Lossy compression scheme, this will avoid degradation of image quality.

Use MED_DCM_DS_update_file() function to update DataSet in a DICOM file, without changing its pixel data.

Critical Data Elements

In this manual, we use the term "Critical Data Elements" for those DEs that are used to help load a DICOM image correctly. It's important to note that when certain IP methods are called, the toolkit alters the Data Set. For example, when an aspect of the image has been altered (for instance, by image processing) so, that the Critical DEs no longer correctly describe the image, the Data Set have to be altered. When the image is actually saved, ImageGear also analyzes the DIB and set its own values for the Critical Data Elements of the file being saved. The original values are ignored.

The Critical Data Elements are:

(0028,0010)	Rows
(0028,0011)	Columns
(0028,0100)	Bits Allocated
(0028,0101)	Bits Stored
(0028,0102)	High Bit
(0028,0004)	Photometric Interpretation
(0028,0103)	Pixel Representation
(0028,0002)	Samples per pixel
(0028,0006)	Planar Configuration
(7FE0,0010)	Pixel Data

PixelData DE does not contain any actual data, it is always empty. The actual pixel data is stored in ImageGear's

DIB, to which the DataSet is attached. Value Representation and Value Length of the Pixel Data tag correspond to VR and VL of PixelData tag, if the DIB is saved to an uncompressed image.

1.2.4.9.2.4.5 Working with DICOM Data Dictionary

The MD component encapsulates the DICOM Data Dictionary (Part 6) in a static and internal table, which the component uses to look up Tags as they are being decoded. It can be used also by your application to provide the DICOM given name or other information about a given Tag. To retrieve this information use the MED_DCM_util_tag_info_get() function. Tag IDs are listed in the enumIGMedTag enumeration.

The Data Dictionary is also used to check the VR of Data Elements as they are being decoded. If a file is Explicit VR then as each Data Element is read, the VR found in the file is matched against the VR in the Data Dictionary. If it is determined that the VR is not appropriate for the Tag type, an error condition may occur.

You can add your own user-defined Tags to the Data Dictionary, or add new Tags that have been added to the DICOM specification so that you keep your application current. The Tags that you add, whether they are user-defined, or new to the DICOM specification, will be stored in a separate table in memory. The enumIGMedTag enumeration is reserved for public DICOM Tags. However, when your application is running, ImageGear Medical will treat the two tables as one, and will be able to process any valid Tag number that you reference. To add a Tag, use MED_DCM_util_tag_info_add().

Since the internal Data Dictionary tables are statically defined, any Tags you wish to add should be added to your initialization code since the table need to be updated each time the toolkit is started up.

The internal Data Dictionary holds several pieces of information for each Tag: it holds the VR, the VM, the version in which the Tag is last used, and the character string name of the Tag as it appears in the DICOM Standard.

The version information is stored because there are many Tags in the DICOM 3 Standard Data Dictionary that are now obsolete or "Retired" as DICOM calls them. If a Tag has been Retired in DICOM 3, then the version is stored as 2. If the Tag has not been retired then it is stored as 3 (for DICOM version 3). Retired Tags also include the string "(RET)" at the end of their name string.

1.2.4.9.2.5 Working with Presentation State Objects

Presentation State (PS) objects serve for the following purposes: consistent display of images on various devices and media, storage of specific settings for display (contrast transformations, geometric transformations), and storage of annotations. They also allow for special display of multi-frame images.

Presentation State object files do not include actual images, but reference one or more images.

Presentation State object files, in addition to LUTs, display parameters and annotations, contain other tags, such as Patient/Study/Series info, Referenced Image UIDs, etc. ImageGear provides read/write access to these tags.

This section provides information about the following:

- Consistent Display of Images
- Grayscale Contrast Transformations
- Geometric Transformations
- Working with Presentation State Objects

Consistent Display of Images

DICOM standard introduces the Standardized Display System. A Standardized Display System may be a printer, a monitor or some other display device which has been calibrated according to the Grayscale Standard Display Function (GSDF). The main feature of such display system is that throughout its display range, equal differences in digital input values correspond to visually equal differences in luminosity. Two Standardized Display Systems will always show the same detail in an image, even if their physical characteristics are different.

The values that can be used as input to a Standardized Display System are called "Presentation Values" ("P-Values"). To map image pixel intensities into P-Values, Presentation Look-Up Table ("P-LUT") is used. It is applied after Modality and VOI LUTs.

Presentation LUT is stored in a Presentation State DataSet.

If a display device is not physically calibrated to comply with GSDF, but its characteristic curve (a table that lists luminosities for each digital input value) is known, it can be calibrated at the software level. In that case, P-Values shall be used as input to a GSDF LUT, which will map them to the device's input values according to GSDF. Thus, a non-standardized device together with its GSDF LUT can be considered as Standardized Display System.

ImageGear Medical component allows you to build a GSDF LUT from a device's Characteristic Curve.

ImageGear uses all available LUTs (Modality LUT, VOI LUT, Presentation LUT and GSDF LUT), to build the general 16x8 or 8x8 LUT that maps image pixel values into display input values.

Grayscale Contrast Transformations

Presentation State DataSet may include VOI and Modality LUTs. Their usage is the same as in normal DICOM images. If either of these LUTs is present, it overrides the LUT found in the image.

Geometric Transformations

Presentation State DataSet may include Rectangle of Interest, scale mode (True size, Scale to Fit, Magnify), and orientation (Rotate, Flip).

Working with Presentation State Objects

You can load Presentation State files in the same way as you load normal DICOM images. However, since PS files do not contain an image, HIGEAR will be set to an empty image (DIB.biCompression |= IG_BI_EMPTY). Data Set of such image is accessible through Medical API. Such images can also be saved in the same way as normal DICOM images.

A HIGEAR containing a Presentation State DataSet can be "applied" to another HIGEAR that contains an image. This operation applies display and annotation settings from Presentation state onto the target image. In the opposite way, display settings and annotations can be exported from an image into another HIGEAR.

When you apply Presentation State to an image, ImageGear updates its 16x8 or 8x8 LUT using the Presentation LUT. However, it does not store Presentation LUT with that image. If you would like to work with Presentation LUT, you should allocate memory for it in your application. Pass Presentation LUT as a parameter to medical display functions. You can also save Presentation LUT to a Presentation State data set.

If a Characteristic Curve is available for a display, you can build a GSDF LUT from it, and use it in calls to medical Presentation State and Display functions. Otherwise, pass a NULL to GSDF LUT parameter.

1.2.4.10 Library Utility Functions

ImageGear's small group of functions, called "library utility" functions, are provided so that you can conveniently obtain the ImageGear version and function status. The functions in this group have names beginning with IG_version_ ...() , IG_error_ ...(), IG_err_ ...().

For instructions on how to use these functions, see the section entitled Detecting and Handling Errors. For information on checking the ImageGear Version, see Checking the ImageGear Version. Detailed information on these functions is provided in Core Component API Reference.

1.2.4.10.1 Checking the ImageGear Version

ImageGear provides two IG_version_ ...() functions:

- IG_version_numbers gives you update information:

```
INT     nVersionMajor;
INT      nVersionMinor;
INT      nVersionUpdate;IG_version_numbers ( &nVersionMajor, &nVersionMinor, 
&nVersionUpdate);
```

- IG_version_compile_date allows you to obtain the compilation date of the ImageGear that you are using:

```
LPSTR             lpszCompileDate;
lpszCompileDate  =  IG_version_compile_date ( void );
```

Upon return from the above, lpszCompileDate will contain a pointer to a string of the form "Mmm dd yyyy," such as "May 09 2011."

1.2.5 Creating Your Imaging Application

This section shows you how to compile and link your program and discusses the final steps in preparing your application for the end user. Because all of the imaging functionality you incorporate for your end user is contained in a single library, this is actually quite easy.

This section provides information about the following:

- Compiling and Linking
  - Creating the Project
  - Project Settings
  - Adding Project Files
- Preparing Your Application for the End User

Refer to the Getting Started chapter to learn the first basic steps of creating your application.

Also, the Using ImageGear chapter provides information about how to define and reference ImageGear's data types, structures, and constants, how to call ImageGear functions, and how to detect and handle errors in order to debug your application.

1.2.5.1 Compiling and Linking

Mac-based applications that call ImageGear functions may be built and compiled using Xcode integrated development environment (IDE) of version 4 and later.

This section provides a step by step description of how a simple sample project can be built. There are many ways to build a project, therefore please use this instruction as a guide but not as the only way for creating an application.

The following Xcode project settings will be required for all applications using the ImageGear Library.

- Creating the Project
- Project Settings
- Adding Project Files

1.2.5.1.1 Creating the Project

1. Start up the Xcode IDE environment.
2. Choose **File > New > Project**. This will bring up the project templates window.
3. From the list of available templates, choose the **Cocoa Application** item and click **Next**.
4. When prompted for the name and other properties of the project that is to be created, supply this information and click **Next**.
5. Choose the destination location of the project.
6. For this example we have chosen the **Samples/Xcode** folder created during the installation of ImageGear software as our destination folder for the new project with a name of **AccuTest**. Once you have saved, a project window with the name of your project will be created.

1.2.5.1.2 Project Settings

Although you have many options when creating a new project, in our example we select only a few.

1. Click the **Targets** tab and then click your target, **AccuTest**.
2. The **Target Editor** is opened, and you can choose the options of selected project.

Below you can see the list of options and their settings, which will help the sample application to work correctly under Mac OS X:

Build Settings Tab, Framework Search Paths

```
/Library/Frameworks
"$(SRCROOT)/../../../Bin"
```

Build Settings Tab, Preprocessor Macros

```
macintosh
_MAC64
IG_PLATFORM_MAC_OS_X
```

1.2.5.1.3 Adding Project Files

Once the project settings have been chosen, you can add the files to our project. Please remember that we are trying to build the sample application project, so our project is being built from the **Samples/Xcode** sub-folder created during the installation process.

For an extended tutorial of Mac OS X development, please refer to Apple documentation, for example, https://developer.apple.com/library/mac/referencelibrary/GettingStarted/RoadMapOSX/books/RM_YourFirstApp_Mac/Articles/GettingStarted.html

The project that Project Assistant creates is a complete Mac OS X application that contains all necessary files organized into several groups. In order to add ImageGear functionality to the sample, you can update existing ImageGearDemo sample code (using copy/paste operations) or just create the necessary files in your project and copy the content of the corresponding files from the ImageGearDemo sample. You also need to include Library/Frameworks/ImageGear18.framework to your project.

If you need PDF support in your application, do the following:

1. Open the "Build Phases" tab in Target settings.
2. Add "Copy Files" build phase.
3. Set destination to "Frameworks".
4. Move all DL*.framework and ICU*.framework files from /Accusoft/ImageGear18/Bin installation directory there.

1.2.5.2 Preparing Your Application for the End User

Removing Your Debugging Error Messages

The section Error Detection and Handling shows you how to produce reports of ImageGear errors to the debug console of Xcode debugger to facilitate the debugging of your application. However once your application is debugged and you are in the final stage of preparing your application for the end user, you would normally remove the error reporting that has been intended for your application debugging. In the final version of your application ready for distribution, all error reports that can appear should be meaningful for the intended end-user.

Excluding Files Licensed Only for Your Own Use

Please remember that in distributing an application using ImageGear functions, you are authorized to distribute only **Library/Frameworks/ImageGear18.framework** and files that are located in **Accusoft/ImageGear18/Bin** directories installed on your computer. You have to remove any other Accusoft ImageGear files that may have been included in your application project while developing it from the final programs or kit that you distribute.

You are permitted to include in your application source code from the sample source programs provided for you in directory **Accusoft/ImageGear18/Samples** and its subdirectories, but you are not permitted to distribute the sample programs themselves.

Providing the Finalized Application's Link to the Shared Library

If you are going to distribute your application created using ImageGear18 framework, you must install **ImageGear18.framework** in **/Library/Frameworks/** directory where it will be found by the application. You also have to distribute the deployment version of **accusoft.<solution name>.imagegear** license file with it. If you don't have an Accusoft ImageGear deployment license, please contact Accusoft to purchase it.

1.2.6 File Format Reference

This chapter provides information about the image formats that ImageGear supports. Before you begin, please refer to the "Encoding vs. Compressing" section below to familiarize yourself with the terms used in this Chapter.

Then you can read the Format Suitability at a Glance section, which briefly delineates which formats best support various types of images. This section provides you with a starting point for deciding which file formats to use in your application.

The section ImageGear Support for Graphics File Formats describes the types of imaging file formats supported by ImageGear and provides useful information about support for some specific formats:

- Support for Adobe PDF/PS Formats
- Support for DICOM File Format
- Support for Metafile Formats
- Support for Multi-Page File Formats

ImageGear Supported Bit Depths section describes the bit depths, and the read/write capabilities of the supported formats. Using this table you can easily find out whether an image can be converted to a particular format.

Detailed information about every ImageGear supported imaging file format or compression can be found in the following sections:

- ImageGear Supported Compressions Reference- here you will find information for every ImageGear supported imaging compression.
- ImageGear Supported File Formats Reference- provides you with the detailed information about every ImageGear supported format, its ID, versions, encoding type, multi-page and alpha channel support, supported compressions, color spaces and bit depths for read and write, as well as information about ImageGear supported features and filter control parameters.

If you are going to use the ImageGear alpha channels or transparency support, please review the section ImageGear Alpha Channel Support or ImageGear Transparency Support.

The section ImageGear Supported Non-Image Data provides detailed information about ImageGear supported metadata.

Encoding vs. Compressing

These two words are often used interchangeably in discussions of graphics file formats. Encoding is actually a broad term under which compression falls. For the sake of clarity, use these terms separately with the following intended meanings:

- Encoding - The manner that data is stored when uncompressed (binary, ASCII, etc.), how it is packed (e.g., 4-bit pixels may be packed at a rate of two pixels per byte), and the unique set of symbols used to represent the range of data items.
- Compressing - A "physical" rewriting of the graphics data so that it is represented by a smaller set of data.

1.2.6.1 Format Suitability at a Glance

The following table contains the formats that are considered most practical for the listed type of data, and can be used as a starting point. You can use a format with the appropriate character depth and one that provides either the most efficient use of space, or the fastest loading and saving capabilities.

Bear in mind that the following table provides recommendations only. To make an informed choice, read about each format in more detail in the section ImageGear Supported File Formats Reference.

Image Type	**Recommended**
Colors	• JPEG • TIFF • PNG
Grayscale/many shades	• TGA • TIFF • JPEG • DICOM
Monochrome / high resolution	• TIFF (CCITT Group 3 compression) • TIFF (CCITT Group 4 compression)
Banking data	• IBM IOCA
Noisy	Color: • PNG • TIFF • JPEG Bi-tonal: • Group 3 • Group 4
Lossless	• GIF • TIFF • PNG • Group 3 • Group 4 • JPEG (Lossless JPEG compression)
Lossy	• TIFF (JPEG compression) • JPEG
Extra data to store	• TIFF • EXIF-JPEG • EXIF-TIFF
Iconic images	• ICO
Facsimiles	• Group 3 • Group 4
Multimedia	• PNG • GIF • JPEG • AVI • QuickTime
Photographic images	• Adobe PSD

	• Adobe PSB • PCD • JPEG
Internet images	• GIF • JPEG • PNG • WBMP
Mac paint programs	• MAC • MAC PICT
PC paint programs and/or graphics arts	• PCX • GEM • WMF • TIFF • EPS • MAC PICT • BMP
X Windows	• XBM • XPM • XWD
Medical data	• DICOM
CAD/Vector	• EPS • WMF
Document processing	• TIFF • Adobe PDF • TXT (ASCII Text)
Gigabyte-sized images	• Adobe PSB • TIFF

1.2.6.2 ImageGear Support for Graphics File Formats

Every application that deals with images has specific kinds of data to store and interchange, from icons to photographs. The various hardware devices used to record and store the graphics data, and the hardware intended to display or print the data also affect the design of the format. This leads to a diversity of file formats. To add to this diversity, different groups of people have different ideas about how to structure and access an image, and what kind of additional information should be stored with an image. Even formats designed to store the same kind of data can differ. Other factors that affect the outcome of the design include memory considerations, storage size, accuracy, and portability.

National and international standardization organizations, such as the American National Standards Institute (ANSI) and the International Standards Organization (ISO), seek to create standards of storage for graphics data. One example is the Joint Photographic Experts Group's (JPEG) creation of the JPEG file format. Some of its intended goals were good image quality, user-chosen compression ratio, and cross-platform flexibility. When an image is called a JPEG, it is assumed to precisely follow the standardized JPEG format.

Formats known as "de facto standards" are those that begin as proprietary formats, but by the forces of the market and sometimes by good quality, become widely supported. Some examples of de facto standards are BMP, GIF, and PCD.

A third group of file formats falls somewhere between officially recognized standards and the strictly proprietary formats. These formats are created by groups of individuals with a common interest who come together to form a more unofficial standards organization. These formats are usually intended to provide an end-all industry standard so that data with the same or similar origins can be shared across different applications or platforms. One example is the TIFF format. TIFF was designed by eight computer technology companies (headed by Aldus Corporation) with the common goal of providing a standard format for storing scanned images.

ImageGear supports graphics file formats from all of these genres, providing you with a complete range of imaging capabilities, including the capture and processing of scanned images.

In addition, ImageGear functionality enable you to exchange data easily from one format to another, and to make improvements in images using powerful image-processing API.

ImageGear list of supported formats includes all of the popular formats, including recognized standards that best utilize the latest imaging technologies.

This section provides information about the following:

- Support for Adobe PDF/PS Formats
- Support for DICOM File Format
- Support for Metafile Formats
- Support for Multi-Page File Formats

See Also:

ImageGear Supported Compressions Reference

ImageGear Supported File Formats Reference

1.2.6.2.1 Support for Adobe PDF/PS Formats

ImageGear provides comprehensive support for Adobe PDF format using its ImageGear PDF Component.

To use support for PDF format as well as manipulate and transform PDF images, the ImageGear PDF Component should be attached to Core ImageGear.

PostScript format is not supported on MacOS X platform.

See Also:

Using ImageGear PDF Component

PDF Component API Function Reference

1.2.6.2.2 Support for DICOM File Format

ImageGear provides comprehensive support for DICOM formats using its ImageGear Medical Component.

To use the support for DICOM format as well as manipulate, transform and process DICOM images ImageGear Medical Component should be attached to Core ImageGear.

See Also:

Using ImageGear MD Component

MD Component API Function Reference

1.2.6.2.3 Support for Metafile Formats

Metafiles contain vector (or geometric) specifications and bitmap pixel data. Vectors, in the realm of computer graphics, define shapes and location of shapes in terms of their relative location within the page.

ImageGear supports the following formats that contain vector data:

- WMF
- WPG
- EPS
- MAC PICT

For all formats, except WMF, the vector data is ignored; only the bitmap data is read and/or written.

When ImageGear loads a Windows Metafile (WMF), it automatically converts each vector specification to bitmap data. The shapes and lines declared by the vector specifications appear in the image, but the original vector instructions are not saved.

1.2.6.2.4 Support for Multi-Page File Formats

ImageGear provides multi-page support for the following file formats:

- Adobe PDF
- AVI
- BMP (OS/2 BMP only)
- CUR
- DCX
- GIF
- IBM AFP
- IBM IOCA
- IBM MO:DCA
- ICO
- IFF
- PCD
- TIFF
- TXT (ASCII Text)

Please see the section Working with Multi-Page Documents to learn how to use ImageGear multi-page functionality.

To create multi-page files, simply save to an existing Adobe PDF, DCX, DICOM, GIF, IBM AFP, orTIFF file. If the file exists, the new page is appended to the file.

It is important to note that ImageGear treats the first page of a multi-page file as page number 1 (not 0).

1.2.6.3 ImageGear Supported Bit Depths

You may want to convert a file from one format to another. You can do this by saving the original file to its desired format by setting the nFormatType parameter to the appropriate value in the saving function. For more information on converting issues, see the sections Loading Images and Saving Images.

When saving a file to another format, remember to ensure that the desired file format is supported for that image. For example, you cannot convert an 8-bit DCX file to an 8-bit CAL file, because the CAL format does not support 8-bit files.

In the tables on the following pages, the columns underneath the numbers indicated size in bits that can actually be saved (written). The columns underneath the letters "R" and "W" indicate that the ImageGear reads and writes, respectively, the corresponding file format.

Format	**R**	**W**	**1**	**4**	**8**	**9-16G**	**24**	**32**	**36**	**48**	**64**
Adobe DNG	+					+				+	
Adobe PDF	+	+	+	+	+		+	+			
Adobe PSB [1]	+	+	+	+	+	+	+	+		+	+
Adobe PSD [2]	+	+	+	+	+	+	+	+		+	+
AFX	+						+				
AVI	+		+	+	+		+				
BMP	+	+	+	+	+	+	+	+[3]			
BTR	+	+	+								
CAL	+	+	+								
CLP	+	+	+	+	+		+				
CUR [4]	+	+	+	+	+			+			
CUT	+	+			+						
DCX	+	+	+	+	+		+				
DICOM	+	+	+		+	+	+	+		+	
EPS [7]	+	+	+	+	+		+				
EXIF-JPEG	+	+			+	+	+	+	+		
EXIF-TIFF	+	+	+	+	+	+	+	+	+	+	+
GEM	+		+	+	+						
GIF	+	+	+	+	+						
Group 3	+	+	+								
Group 3 2D	+	+	+								
Group 4	+	+	+								
IBM AFP	+		+	+	+		+				
IBM IOCA	+	+[8]	+	+	+		+				
IBM MO:DCA	+	+[9]	+	+	+		+				
ICO	+	+	+	+	+			+			
IFF	+	+	+	+	+		+				
IMG	+		+		+						
IMR	+		+								
IMT	+	+	+								
JPEG	+	+			+[10]	+	+	+	+	+	
KFX	+		+								

LV	+		+								
MAC	+		+								
MAC PICT	+	+	+	+	+		+				
MSP	+		+								
NCR	+	+	+	+[11]							
PBM	+	+	+		+	+	+			+	
PCD	+						+				
PCX	+	+	+	+	+		+				
PGM [12]	+	+			+	+					
PNG [13]	+	+	+	+	+	+	+	+		+	+
PNM [14]	+	+	+		+	+	+			+	
PPM	+	+					+			+	
QuickTime	+						+				
RAS	+	+	+	+[15]	+		+				
RAW	+		+	+	+	+	+	+			
Scitex CT [16]	+	+						+			
SGI	+	+			+	+	+	+			
TGA	+	+	+		+	+	+	+			
TIFF [17]	+	+	+	+	+	+	+	+	+	+	+
TXT (ASCII Text) [18]	+										
WBMP	+	+	+								
WMF	+	+	+	+	+		+				
WPG	+		+	+	+						
XBM	+	+	+								
XPM	+	+	+	+	+		+[19]				
XWD	+	+	+	+	+		+				

[1] 48- and 64-bit images support is Read only.

[2] 48- and 64-bit images support is Read only.

[3] Read only support.

[4] Supports Extra ((1, 3, 4, 8)*2)-bit images also.

[7] Screen Preview image only, when reading.

[8] Write support for 1-bit images only.

[9] Write support for 1-bit images only.

[10] 8-bit grayscale.

[11] Read only support.

[12] Supports 16-bit grayscale images also.

[13] Supports 48(RGB)- and 64(RGB+alpha)-bit images.

[14] Supports 16-bit grayscale and 48-bit color images also.

[15] Read only support.

[16] The native format is CMYK. With full CMYK support enabled, you can use 32-bit images.

[17] Read only supports also 3- and 6 bpp for RGB and LAB color spaces for Deflate, LZW (Lempel-Ziv-Welch), Packbits compressions and uncompressed.

[18] Converts to raster image when loaded.

[19] 24-bit has Read only support.

1.2.6.4 ImageGear Alpha Channel Support

Alpha channel is an additional image channel that specifies transparency (or opacity) of each pixel in the image. ImageGear provides alpha channel support for the following file format filters:

Format	**Support**
Adobe PSB	Supports single Alpha channel for Read only. Supports additional Alpha channels as ImageGear Extra channels for Read only.
Adobe PSD	Supports single Alpha channel for Read only. Supports additional Alpha channels as ImageGear Extra channels for Read only.
CUR	Supports single 8-bit alpha channel for read/write. Supports single 1-bit alpha channel for read only.
EXIF-JPEG	Supports single Alpha channel for Read/write.
EXIF-TIFF	Supports single Alpha channel for read only. Supports additional Alpha channels as ImageGear Extra channels for read only.The following compressions are supported with alpha channel: • Uncompressed • Packed Bits • LZW • Deflate
ICO	Supports single 8-bit alpha channel for read/write. Supports single 1-bit alpha channel for read only.
JPEG	Supports single Alpha channel for Read/write.
MAC PICT	Supports single alpha channel for read only. Alpha channel have to be 8-bit image.
PNG	Supports single Alpha channel for read/write.
RAW	Supports single Alpha channel in uncompressed images for read only.
SGI	Supports single alpha channel for read and write.
TGA	Supports single alpha channel for read and write.
TIFF	Supports single Alpha channel for read and write. Supports additional Alpha channels as ImageGear Extra channels for read and write.The following compressions are supported with alpha channel: • Uncompressed • Packed Bits • LZW • Deflate

1.2.6.5 ImageGear Transparency Support

Transparency allows you to specify a palette index or a color to be transparent. When the image is displayed, all pixels having this index (color) show background rather than the pixel's color.

Transparency requires very little storage space in the file - only a few bytes. However, the pixels can't be semi-transparent; also, transparent color occupies a palette entry or a color, which reduces the possible range of colors for the image. Alpha channel requires significantly more storage space in the file, but it does not have the limitations listed above.

ImageGear provides transparency support for the following file format filters:

Format	**Support**
GIF	Supports transparency for Read and Write.
PNG	Supports transparency for Read and Write.
XPM	Supports transparency for Read and Write.

1.2.6.6 ImageGear Supported Compressions Reference

All imaging files compressions can be divided on three basic types:

- One-dimensional compression - the raster data is treated as one continuous data stream. Each byte read is compared to the previous byte. This compression method is not concerned with delineating lines of data.
- 2D compression can be thought of as "Differencing Compression", where the data stored is a representation of the differences in data values from previous data values. In 2-D compression, the encoding of one line is determined by the contents of the previous line. This method of compression is best used for black-and-white images where the black pixels tend to fall into groups.
- 3D compression is a new branch of data compression aimed at the 3D models and other geometric datasets used in computer graphics, virtual reality, video games, CAD/CAM, and many scientific, engineering, and medical applications.

Existing 3D compression algorithms use both techniques adapted from the 1D and 2D cases (like wavelets, entropy coding, and predictive coding), and completely different approaches that take advantage of the properties of 3D surfaces (like Edgebreaker, Subdivision Surfaces, and triangle strips).

ImageGear supports the following compressions:

- ASCII
- CCITT Group 3
- CCITT Group 3 2D
- CCITT Group 4
- Deflate
- Huffman
- IBM MMR
- JPEG
- Lossless JPEG
- LZW (Lempel-Ziv-Welch)
- Packbits
- Progressive JPEG
- RAW
- RLE

1.2.6.6.1 ASCII

Full Name	American Standard Code for Information Interchange (ASCII)
Compression ID	IG_COMPRESSION_ASCII = 23
ImageGear Component	Core
Bit Depth	Gray level: 1, 8, 16 bpp; RGB 24, 48 bpp
File Formats	PBM, PGM, PNM, PPM
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Acronym for the American Standard Code for Information Interchange. ASCII is a code for representing English characters as numbers, with each letter assigned a number from 0 to 127. For example, the ASCII code for uppercase M is 77. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

Text files stored in ASCII format are sometimes called ASCII files. Text editors and word processors are usually capable of storing data in ASCII format, although ASCII format is not always the default storage format. Most data files, particularly if they contain numeric data, are not stored in ASCII format. Executable programs are never stored in ASCII format.

1.2.6.6.2 CCITT Group 3

Full Name	CCITT Group 3
Compression ID	IG_COMPRESSION_CCITT_G3 = 3
ImageGear Component	Core
Bit Depth	1
File Formats	BTR, EPS, Group 3, IBM IOCA, IBM MO:DCA, IMT, LV, NCR, Adobe PDF, TIFF, RAW
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

This is a 1-D version of the CCITT Group 3 compression scheme. It uses a static table of values to assign codes to run lengths. Frequently occurring run lengths are given smaller codes. (The most frequent are usually black runs of 2 or 4 pixels).

1.2.6.6.3 CCITT Group 3 2D

Full Name	CCITT Group 3 2D
Compression ID	IG_COMPRESSION_CCITT_G32D = 5
ImageGear Component	Core
Bit Depth	1
File Formats	BTR, EPS, Group 3 2D, IBM IOCA, IBM MO:DCA, Adobe PDF, TIFF, RAW
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

This is a 2-D version of the CCITT Group 3 compression scheme. It includes an error recovery algorithm for error transmissions; an error in one line does not translate to garbage output for the rest of the file. All modern fax machines support this format.

1.2.6.6.4 CCITT Group 4

Full Name	CCITT Group 4
Compression ID	IG_COMPRESSION_CCITT_G4 = 4
ImageGear Component	Core
Bit Depth	1
File Formats	CAL, EPS, Group 4, IBM IOCA, IBM MO:DCA, IMR, IMT, KFX, LV, NCR, Adobe PDF, TIFF, RAW
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

The G4 compression is two-dimensional by default. It is very similar to the G3 2D compression, but it can produce compressed images that are half the size of a G3-compressed file.

It is slower, however, and does not have the same error recovery built in that the G3 format has. The decrease in speed occurs because G4 was designed specifically for encoding disk data. For this reason, it may be advisable to use the G3-Fax compression scheme if final compression size is not crucial, but speed is.

1.2.6.6.5 Deflate

Full Name	Deflate compression
Compression ID	IG_COMPRESSION_DEFLATE = 14
ImageGear Component	Core
Bit Depth	1, 2, 4, 8, 16 bpc
File Formats	Adobe PDF, PNG, TIFF
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

"Zip-in-TIFF" compression. Deflate compression, sometime known as "zip" compression, uses another variant of the LZW compression method and so gives similar results, but is not restricted by any licenses.

1.2.6.6.6 Huffman

Full Name	Huffman encoding
Compression ID	IG_COMPRESSION_HUFFMAN = 2
ImageGear Component	Core
Bit Depth	1, 24
File Formats	NCR, PCD, TIFF
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Developed in 1952 by David Huffman, this is one of the older compression methods. The encoding and decoding processes are complex relative to today's standards, but the compression ratio can be high if the image contains many repeat data values. It is best used for images with little or no pixel noise, e.g. cartoons or drawings with large areas of the same color and intensity (like a monotone sky). This compression scheme is often used by other compression algorithms for extra compression.

The Huffman method uses a conversion table to assign codes for each value, based on frequency of occurrence. The file is scanned for all of its values, with the values and their frequency of occurrence tallied. Using a binary tree, values are paired off by frequency of occurrence, beginning with the least frequent values. As the tree progresses upward, the least occurring values at the bottom of the tree continue to be incremented a bit at a time, with one bit added for each new branch added to the tree. In the end, the values that occur the most (at the top of the tree) have the shortest codes.

A potential problem with this compression method is decoding; the file's variable-length codes can cause the dropping or adding of a bit to the end of a line, thereby throwing off subsequent lines of data.

1.2.6.6.7 IBM MMR

Full Name	IBM Modified Modified Read
Compression ID	IG_COMPRESSION_IBM_MMR = 15
ImageGear Component	Core
Bit Depth	1
File Formats	IBM IOCA, IBM MO:DCA
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Compression for black and white documents, similar to CCITT Group 4.

1.2.6.6.8 JPEG

Full Name	JPEG compression
Compression ID	IG_COMPRESSION_JPEG = 6
ImageGear Component	Core
Bit Depth	Gray level: 8, 16 bpp; RGB: 24, 36 bpp; CMYK: 32 bpp; RGB+Alpha: 32 bpp
File Formats	AFX, AVI, JPEG, DICOM, Adobe DNG, EPS, EXIF-JPEG, MAC PICT, Adobe PDF, TIFF
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

This file compression method obtains a high compression ratio when used with detailed photographic images (its intended use). It is not a good compression choice for images with a small number of colors and high contrast edges, or for color-mapped data. Part of JPEGs success in high compression is due to the fact that it is a "lossy" compression method, meaning that the compression results in the loss of some data that is determined to be unimportant or unnecessary. This does not necessarily result in a visible reduction of image quality.

JPEG is highly flexible - it allows you to make a "quality" setting that determines the amount of loss that occurs and affects the size of the resulting compressed file.

The JPEG algorithm takes into account that the human eye is more sensitive to changes in brightness than to number of colors. Rather than saving the color data from each pixel in an image, it saves information on the rate of change of color, or "frequency information." More loss is allowed in the color data than in the brightness data. Some of the compression of the color is achieved by converting the RGB values to YCbCr color scheme. ImageGear supports two other JPEG compression modes--Lossless JPEG and Progressive JPEG.

See Also:

Lossless JPEG, Progressive JPEG

1.2.6.6.9 Lossless JPEG

Full Name	Lossless JPEG compression
Compression ID	IG_COMPRESSION_JPEG = 6
ImageGear Component	Core
Bit Depth	Gray level: 8, 16 bpp; RGB: 24, 48 bpp
File Formats	DICOM, JPEG, TIFF
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Lossless JPEG is an extension to the normal JPEG standard. One of the main algorithmic differences between the two is that the lossless JPEG does not apply a Discrete Cosine Transform. Rather, it uses a Predictive scheme. For each pixel, the values of one or several neighboring pixels are added to the value of the original pixel and then subtracted from the value of the original pixel. This method yields smaller values that require fewer bits per pixel to store.

ImageGear allows you to set the number of neighboring pixels to use in calculating the "predictor value".

1.2.6.6.10 LZW (Lempel-Ziv-Welch)

Full Name	Lempel-Zif-Welch (LZW) compression
Compression ID	IG_COMPRESSION_LZW = 8
ImageGear Component	GIF/TIFF-LZW
Bit Depth	1, 4, 8, 16 bpc
File Formats	GIF, Adobe PDF, TIFF
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

LZW compression works by finding patterns of data and assigning codes. It works best on highly-patterned images. Images with irregular patterning, or "noise," are not good candidates for this type of compression.

This compression scheme is "dictionary-based". This refers to the array of codes that identify each data pattern found in the image. The "dictionary" begins with a table that contains a code for each possible value in the image. If LZW compression is used on 8-bit images, a LZW "dictionary" is initialized with codes for 256 (28) values. As the file data is read, new values are added to the table for each unique pattern of data found. In the interest of saving space, the dictionary is not saved with the compressed file. The same dictionary is actually rebuilt when the data is decoded.

1.2.6.6.11 Packbits

Full Name	Packed bits compression
Compression ID	IG_COMPRESSION_PACKED_BITS = 1
ImageGear Component	Core
Bit Depth	1, 4, 8, 16 bpc
File Formats	Adobe PSD, TIFF
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Packbits compression seeks repeated data values. Packbits is considered an RLE (run-length encoding) compression scheme because it looks for "runs" or repeated values, and tallies their number, or "length". While its name implies that runs of bits are "packed" together, it is actually runs of bytes. It is very similar to the Macintosh Packbits compression used by Macpaint, except that the Packbits compression used for a TIFF allows the dimensions of the image to vary.

Packbits works by reducing repeated strings of the same characters into two components: the "run count" and the "run value". The count and value are stored in one byte each. Each two-byte grouping is referred to as an RLE packet. It is not a good compression scheme for images with large color ranges, as these do not tend to have many runs of the same color.

The terms "RLE" and "Packbits" are often used synonymously.

1.2.6.6.12 Progressive JPEG

Full Name	Progressive JPEG compression
Compression ID	IG_COMPRESSION_PROGRESSIVE = 17
ImageGear Component	Core
Bit Depth	Gray level: 8, 16 bpp; RGB: 24, 36 bpp; CMYK: 32 bpp; RGB+Alpha: 32 bpp
File Formats	JPEG
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Progressive JPEG is considered an extension to the JPEG standard. It produces the same kind of lossy compression as normal JPEG compression (see above), except that it saves multiple copies of the same image using different levels of quality. There is no hard limit on the number of "scans" that may be stored.

When a Progressive JPEG-compressed image is decompressed, the filter decompresses the lowest quality image first. This can be helpful for quickly displaying a version of the image that you are loading. The benefit of this compression is the fast display of an image that is recognizable. The downside is that a JPEG decompression is performed more than once.

1.2.6.6.13 RAW

Full Name	RAW compression (Uncompressed binary compression)
Compression ID	IG_COMPRESSION_RAW = 24
ImageGear Component	Core
Bit Depth	Gray level: 1, 8, 16 bpp; RGB 24, 48 bpp
File Formats	PBM, PGM, PNM, PPM
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

Uncompressed binary compression. PBM/PGM/PNM/PPM formats use the term "RAW" for uncompressed binary compression, as opposed to ASCII compression.

1.2.6.6.14 RLE

Full Name	Run length encoding compression
Compression ID	IG_COMPRESSION_RLE = 7
ImageGear Component	Core
Bit Depth	1, 4, 8, 16 bpc
File Formats	BMP, CLP, CUT, DCX, DICOM, GEM, IFF, MAC, MSP, PCX, Adobe PDF, RAS, SGI, TGA, WMF, WPG
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

Comments:

RLE (Run Length Encoding) is normally a 1-dimensional compression scheme. Working sequentially from left to right and top to bottom, it compares the value of each byte with the value of the previous byte. Each data value is recorded into a "packet" of two bytes where the first byte contains the number of times the value is repeated, and the second packet contains the actual value. The bytes in the pocket are called the "run count" and the "run value". When an image contains many repeat values, the compression ratio is very high (for example, if every byte in a 100 byte image were the same, its size could be reduced to 2 bytes giving a 50:1 ratio). A very noisy image, or a plain ASCII text file typically does not compress well, and in fact could become larger for example, if all bytes in image are different from the ones next to them, the image doubles, because 2 bytes are used to store each byte in the image.

The terms "RLE" and "Packbits" are often used synonymously.

1.2.6.7 ImageGear Supported File Formats Reference

This section represents the reference for every imaging file format supported by ImageGear.

The following are the characteristic features for file format description:

- Full Name - Full name of the File Format
- Format ID - ImageGear constant that determines the File Format ID used for loading this file in ImageGear-based application. See the section Working with Format Filters.
- File Extension(s) - The used extensions of the File Format
- Data Type - The type of the File Format (raster, vector, metafile)
- Data Encoding - The type of data encoding. Please see the section Encoding vs. Compressing.
- ImageGear Multi-Page Support - Shows if ImageGear supports the File Format as multi-page or single-page. Please see also the section Support for Multi-Page File Formats.
- ImageGear Alpha Channel Support - Shows if ImageGear supports the alpha channel for the File Format or not. Please see also the section ImageGear Alpha Channel Support.
- ImageGear Platforms Support - Shows ImageGear platform versions that support the File Format.
- ImageGear Supported Versions - Shows the versions of the File Format supported by ImageGear.
- ImageGear Supported Features - Shows the ImageGear Format Filter features supported by ImageGear for this format. Please see the section Working with Format Filters.
- ImageGear Read Support - Provides all compressions, color spaces, channels and bit depths supported by ImageGear for the file format reading.
- ImageGear Write Support - Provides all compressions, color spaces, channels and bit depths supported by ImageGear for the file format writing.
- ImageGear Filter Control Parameters - Provides all filter control parameters supported by ImageGear for this format filter. Please see also Working with Format Filters.
- Comments - Some general information about format encoding and compression structure.
- References Used - References to the information sources for the File Format.

Currently ImageGear supports the following File Formats:

- Adobe DNG
- Adobe PDF
- Adobe PSB
- Adobe PSD
- AFX
- AVI
- BMP
- BTR
- CAL
- CLP
- CUR
- CUT
- DCX
- DICOM
- EPS
- EXIF-JPEG
- EXIF-TIFF
- GEM
- GIF
- Group 3
- Group 3 2D
- Group 4
- IBM AFP
- IBM IOCA
- IBM MO:DCA
- ICO
- IFF
- IMG
- IMR

- IMT
- JPEG
- KFX
- LV
- MAC
- MAC PICT
- MSP
- NCR
- PBM
- PCD
- PCX
- PGM
- PNG
- PNM
- PPM
- QuickTime
- RAS
- RAW
- Scitex CT
- SGI
- TGA
- TIFF
- TXT (ASCII Text)
- WBMP
- WMF
- WPG
- XBM
- XPM
- XWD

1.2.6.7.1 Adobe DNG

Full Name	Digital Negative file format
Format ID	IG_FORMAT_DNG = 108
File Extension(s)	*.dng
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported Versions:

- Version 1.0.0.0
- Version 1.1.0.0 - fixed incompatibility in JPEG Lossless compression, added new tags

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 9..16 bpp
  - RGB: 24, 48 bpp *
- IG_COMPRESSION_JPEG:
  - Grayscale: 16 bpp
  - RGB: 24, 48 bpp *

* 8bpc DNG loading has been enabled via promoting the image to 16bpc before loading.

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
APPLY_COLORSPACE_CONVERSION	AT_BOOL	TRUE	TRUE, FALSE	Set to TRUE to convert raw image to linear sRGB color space during loading. Has effect only if RECONSTRUCT_COLORS is TRUE.
APPLY_TONE_CORRECTION	AT_BOOL	TRUE	TRUE, FALSE	Set to TRUE to automatically adjust image tone and apply sRGB gamma correction during loading. Has effect only if bothRECONSTRUCT_COLORS and APPLY_COLORSPACE_CONVERSION are TRUE.
RECONSTRUCT_COLORS	AT_BOOL	TRUE	TRUE, FALSE	ImageGear attempts to reconstruct full color image from the camera raw image. Otherwise, ImageGear loads raw pixel data without any processing.

Comments

This file format was developed by Adobe as a non-proprietary format for unified storage of "raw" images from digital cameras. DNG image stores unprocessed pixel data obtained from camera's sensor, and keeps information about color, contrast and brightness adjustments, sharpening, as well as many other parameters, in its tags. This provides greater possibilities for image correction and enhancement, compared to commonly used formats such as JPEG, EXIF or TIFF.

The fact that the format is non-proprietary allows software vendors to provide support for DNG in their applications, with complete control over the image reconstruction process.

DNG extends TIFF/EP format. It adds a set of new tags for parameters that control reconstruction of full color image from the raw data.

References Used

ADOBE SYSTEMS INCORPORATED. Digital Negative (DNG) specification.

1.2.6.7.2 Adobe PDF

Full Name	Adobe PDF (Adobe Portable Document Format)
Format ID	IG_FORMAT_PDF = 56
File Extension(s)	*.pdf
Data Type	Vector Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET

To enable the support of the PDF format, attach the ImageGear PDF Component to Core ImageGear.

ImageGear Supported Versions:

- Adobe® PDF version 1.7
- Adobe® PDF version 1.6
- Adobe® PDF version 1.5

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEWRITEPSUPPORT - multi-page file writing
- IG_FLTR_PAGEDELETESUPPORT - page deleting from multi-page file
- IG_FLTR_PAGESWAPSUPPORT - page swapping in multi-page files

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp.
- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp.
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp.
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp.
- IG_COMPRESSION_CCITT_G32D:
  - Indexed RGB: 1 bpp.
- IG_COMPRESSION_JPEG:
  - Indexed RGB: 8 bpp;
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp.
- IG_COMPRESSION_JPEG2K:

    - Indexed RGB: 1, 4, 8 bpp;
    - Grayscale: 8 bpp;
    - RGB: 24 bpp;
    - CMYK: 32 bpp.
- IG_COMPRESSION_LZW:
    - Indexed RGB: 1, 4, 8 bpp;
    - Grayscale: 8 bpp.

ImageGear Write Support:

- IG_COMPRESSION_NONE:
    - Indexed RGB: 1, 4, 8 bpp;
    - Grayscale: 8 bpp;
    - RGB: 24 bpp;
    - CMYK: 32 bpp.
- IG_COMPRESSION_DEFLATE:
    - Indexed RGB: 1, 4, 8 bpp;
    - Grayscale: 8 bpp;
    - RGB: 24 bpp;
    - CMYK: 32 bpp.
- IG_COMPRESSION_RLE:
    - Indexed RGB: 1, 4, 8 bpp;
    - Grayscale: 8 bpp;
    - RGB: 24 bpp;
    - CMYK: 32 bpp.
- IG_COMPRESSION_CCITT_G3:
    - Indexed RGB: 1 bpp.
- IG_COMPRESSION_CCITT_G4:
    - Indexed RGB: 1 bpp.
- IG_COMPRESSION_CCITT_G32D:
    - Indexed RGB: 1 bpp.
- IG_COMPRESSION_JPEG:
    - Indexed RGB: 8 bpp;
    - Grayscale: 8 bpp;
    - RGB: 24 bpp;
    - CMYK: 32 bpp
- IG_COMPRESSION_JPEG2K:
    - Indexed RGB: 1, 4, 8 bpp;
    - Grayscale: 8 bpp;
    - RGB: 24 bpp;
    - CMYK: 32 bpp.
- IG_COMPRESSION_LZW:
    - Indexed RGB: 1, 4, 8 bpp;
    - Grayscale: 8 bpp.

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
ALLOW_XFA	AT_BOOL	FALSE	TRUE, FALSE	Specifies whether to allow opening PDF documents with XFA content embedded. • TRUE - PDF documents with XFA content will open without any error, but XFA content will not be available or visible.

				• FALSE - (default) PDF documents with XFA content will not open, and an ImageGear error will appear.
DEPTH	UINT	24	1, 8, 24	Specifies bit depth in bits per pixels during the PDF document conversion into the raster image. A higher value indicates a higher quality raster image and a larger amount of memory required for rasterization. The DIB of the output raster image has this value as a bit count.
DRAW_MODE	UINT	1	1, 2	Specifies the default PDF rendering method used by the ImageGear PDF component to draw PDF page content. • 1 - Draws entire page content into the cache. This function is optimized for fast scrolling. • 2 - Draws visible page content area. This function is optimized for fast rendering, but would re-render the content each time it is scrolled.
INC_REND	AT_BOOL	FALSE	TRUE, FALSE	Specifies whether incremental rendering or rendering at once should be performed.
INDEPENDENT_PAGESIZE	AT_BOOL	FALSE	TRUE, FALSE	(Used with PDF write only.) • If this parameter is FALSE, when saving a raster image into the PDF document, the width and height of the

				newly created page is set to the width and height of the previous page in the PDF document. If the previous page does not exist, the width and height are calculated from the image resolution and size as follows: Width = <width of the raster image> * 72 / <X DPI of the raster image> Height = <height of the raster image> * 72 / <Y DPI of the raster image> If the image resolution is not defined, the width and height are set to the width and height of the A4 page, which is 612x792. • If this parameter is TRUE, use the PAGE_HEIGHT and PAGE_WIDTH parameters to set the page size.
PAGE_HEIGHT	UINT	0	Any non-negative value	(Used with PDF write only.) This parameter sets the height of the page (in 1/72 inches). If this option is 0, the height is calculated from the image resolution and size as follows:

				Height = <height of the raster image> * 72 / <Y DPI of the raster image> If the image resolution is not defined, the height is set to the height of an A4 page, which is 792. This option is not used if RESOLUTION_X is FALSE.
PAGE_WIDTH	UINT	0	Any non-negative value	(Used with PDF write only.) This parameter sets the width of the page (in 1/72 inches). If this option is 0, the width is calculated from the image resolution and size as follows: Width = <width of the raster image> * 72 / <X DPI of the raster image> If the image resolution is not defined, the width is set to the width of the A4 page, which is 612. This option is not used if RESOLUTION_X is FALSE.
PASSWORD	LPCHAR	""	Any	Specifies the password string for the password of the protected PDF documents.
PRINT_DEPTH	UINT	8	1, 8, 24	Specifies bit depth in bits per pixels during the PDF document printing. A higher value indicates a higher quality raster image and a larger amount of memory required for printing.
PRINT_RESOLUTION_X	UINT	300	Any positive value, inclusively between 1 and 2147483647	Specifies the horizontal

				resolution in dots per inch during PDF document printing. A higher value indicates a higher quality image to be printed.
PRINT_RESOLUTION_Y	UINT	300	Any positive value, inclusively between 1 and 2147483647	Specifies the vertical resolution in dots per inch during the PDF document printing. A higher value indicates a higher quality image to be printed.
RESOLUTION_3D	UINT	72	Any except 0	Specifies the resolution in dots per inch used for generating a pre-rendered bitmap of the default view of the 3D artwork. Producers should provide bitmaps of appropriate resolution for all intended uses of the document, i.e., a high-resolution bitmap for high-quality printing and a default screen-resolution bitmap for on-screen viewing.
RESOLUTION_X	UINT	72	Any	Specifies the horizontal resolution in dots per inch during the PDF/PS document conversion into a raster image. The higher this value, the higher-quality raster image you get after rasterization. The DIB of the output raster image has this value as an X resolution.
RESOLUTION_Y	UINT	72	Any	Specifies the vertical resolution in dots per inch during the PDF/PS document conversion into a raster image. The higher this value, the higher-quality raster image you get after rasterization. The DIB of the output raster image has this value as an Y resolution.

SAVE_FLAGS	UINT	IG_PDF_OPTIMIZED	A bit composition of an OR of the following values: • IG_PDF_OPTIMIZED = 32 - perform garbage collection on unreferenced objects. • IG_PDF_LINEARIZED = 4 - write the file linearized for page serving over remote connections. • IG_PDF_DONT_SAVE_FILE_ATTRIBUTES = 65536 - prevent the file attributes and security settings of a PDF document opened from an existing PDF file from being copied over when saved to a new PDF file. • IG_PDF_OPTIMIZE_XOBJECTS = 4194304 - merge identical forms and images, as determined by an MD5 hash of their contents.	(Used with PDF write only.) Specifies an option for saving a PDF file that allows you to remove unreferenced objects, often reducing file size, as well as to write a linearized file for page-served remote (network) access.
SAVE_MAJOR	UINT	0	0, 1	Specifies major PDF version number of the document for saving. If major equals 0, both major and minor are ignored and the document is saved to the library's default version. Make sure that the document conforms to the version number you are setting.
SAVE_MINOR	UINT	0	6, 5, 4, etc.	Specifies minor PDF version number of the document for saving. Make sure that the document conforms to the version number you are setting.
SMOOTH_FLAGS	UINT	13	A bit composition of an OR of the following values: • 1 - Draw smooth text • 2 - Draw smooth line art • 4 - Draw smooth image • 8 - Enhance thin lines	Specifies smooth settings for PDF rasterization.
TEXT_ENCODING	UINT	IG_PDF_TEXTENC_NONE	• IG_PDF_TEXTENC_NONE = 1 - no encoding used • IG_PDF_TEXTENC_ASCII_85 = 2 - ASCII 85 encoding used • IG_PDF_TEXTENC_ASCII_HEX = 3 - ASCII HEX encoding used	(Used with PDF write only.) Specifies which encoding scheme should be used to convert binary image data to the text format when saving raster image into a PDF document.
USE_CROP_BOX	AT_BOOL	TRUE	TRUE, FALSE	Specifies whether to use PDF crop box rectangle for page layout. • TRUE - use

		PDF crop box rectangle for page layout • FALSE - use PDF media rectangle for page layout

Comments:

Please see the section Using ImageGear PDF Component.

1.2.6.7.3 Adobe PSB

Full Name	PSB (Adobe Photoshop Big)
Format ID	IG_FORMAT_PSB = 112
File Extension(s)	*.psb
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	Read, Write
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Yes (see Comments for more information).
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 2.0
- Version 2.5
- Version 3.0
- Version 8.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - Grayscale + Alpha + Extra: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp;
  - RGB + Alpha + Extra 8, 16 bpc;
  - CMYK: 32, 64 bpp;
  - CMYK + Extra: 8, 16 bpc;
  - Lab: 24, 48 bpp;
  - Lab + Extra: 8, 16 bpc;
  - HSL: 24, 48 bpp;
  - HSL + Extra: 8, 16 bpc
- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - Grayscale + Alpha + Extra: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp;
  - RGB + Alpha + Extra 8, 16 bpc;
  - CMYK: 32, 64 bpp;
  - CMYK + Extra: 8, 16 bpc;
  - Lab: 24, 48 bpp;
  - Lab + Extra: 8, 16 bpc;
  - HSL: 24, 48bpp;
  - HSL + Extra: 8, 16 bpc

- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - Grayscale + Alpha + Extra: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp;
  - RGB + Alpha + Extra 8, 16 bpc;
  - CMYK: 32, 64 bpp;
  - CMYK + Extra: 8, 16 bpc;
  - Lab: 24, 48 bpp;
  - Lab + Extra: 8, 16 bpc;
  - HSL: 24, 48 bpp;
  - HSL + Extra: 8, 16 bpc

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8 bpp;
  - Grayscale + Alpha: 16 bpp;
  - Grayscale + Alpha + Extra: 8 bpc;
  - RGB: 24 bpp;
  - RGB + Alpha: 32 bpp;
  - RGB + Alpha + Extra: 8 bpc;
  - CMYK: 32 bpp;
  - CMYK + Extra: 8 bpc;
  - Lab: 24 bpp;
  - Lab + Extra: 8 bpc
- IG_COMPRESSION_PACKED_BITS:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8 bpp;
  - Grayscale + Alpha: 16 bpp;
  - Grayscale + Alpha + Extra: 8 bpc;
  - RGB: 24 bpp;
  - RGB + Alpha: 32 bpp;
  - RGB + Alpha + Extra: 8 bpc;
  - CMYK: 32 bpp;
  - CMYK + Extra: 8 bpc;
  - Lab: 24 bpp;
  - Lab + Extra: 8 bpc

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
LOAD_FIRST_EXTRA_CHANNEL_AS_ALPHA	AT_BOOL	FALSE	TRUE, FALSE	This parameter specifies how ImageGear should load the first extra channel of a PSB image. If FALSE, ImageGear loads all extra channels according to their descriptors in the file header. If the channel contains transparency information, ImageGear loads it as Alpha channel. Otherwise ImageGear loads it as Extra channel. If TRUE, ImageGear loads first extra channel as Alpha channel. This mode provides backward compatibility with

				previous versions of ImageGear.
READ_LAYER_INDEX	INT	-1	Any integer, but no more than number of layers	When > -1 specifies a zero based index of layer mask image to load, otherwise reads the composition image.
READ_LAYER_MASK	AT_BOOL	FALSE	FALSE, TRUE	When TRUE PSB layer mask images are loaded as usual pages of multi-page file, otherwise layer masks are ignored.
SAVE_THUMBNAIL	AT_BOOL	FALSE	FALSE, TRUE	Gets/Sets thumbnail flag. If TRUE then thumbnail will be added to image.
THUMBNAIL_ENABLE	AT_BOOL	FALSE	FALSE, TRUE	When TRUE thumbnail reading function loads thumbnail image provided by PSB format.
THUMBNAIL_HEIGHT	UINT	64	Any positive value	Gets/Sets thumbnail height.
THUMBNAIL_WIDTH	UINT	64	Any positive value	Gets/Sets thumbnail width.

Comments:

PSB is a newer version of PSD designed for files over 2 gigabytes, supporting up to 300,000 pixels in any dimension. The PSB format is identical to the Photoshop native format (PSD) in many ways.

The PSD file is considered by many in the computer graphics arts community as an industry standard.

The PSD/PSB is organized into 5 major segments of data: the header, 3 informational blocks, and the bitmap data. The short header always contains a "signature" of 8PPS, as well as these fields: width, height, and bit depth of the bitmap.

The first block of informational data is called the "Color Mode Data Block". It begins with a value for the length of the block. If the image has a palette, it is located here.

The next block is called the "Image Resources Block". Like the previous block, it first gives the length of the block. An ID field is filled with one of many possible values that indicate the structure where the data is stored. It may contain such parameters as resolution.

The last informational block is called the "Layer and Mask Instruction Block". After a value for the length of the block, it tells how many "layer records" follow. There is a record for each layer in the image. Each record begins with a channel ID and the length of the data in the record. After the records, a "Layer Mask" section may be stored, if applicable.

The bitmap data represents the last segment of a PSD/PSB file.

For historic reasons, ImageGear uses the IG_COMPRESSION_PACKED_BITS constant for saving Packed Bits compressed PSD/PSB images, and uses the IG_COMPRESSION_RLE constant to report Packed Bits compression when reading PSD/PSB images. For the PSD/PSB format, the terms "Packed Bits" and "RLE" are used synonymously.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats, 2d ed. Sebastopol, CA: O'Reilly & Associates, Inc., 1996.

1.2.6.7.4 Adobe PSD

Full Name	PSD (Adobe Photoshop)
Format ID	IG_FORMAT_PSD = 36
File Extension(s)	*.psd
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	Read, Write
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Yes (see Comments for more information).
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 2.0
- Version 2.5
- Version 3.0
- Version 8.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - Grayscale + Alpha + Extra: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp;
  - RGB + Alpha + Extra 8, 16 bpc;
  - CMYK: 32, 64 bpp;
  - CMYK + Extra: 8, 16 bpc;
  - Lab: 24, 48 bpp;
  - Lab + Extra: 8, 16 bpc;
  - HSL: 24, 48 bpp;
  - HSL + Extra: 8, 16 bpc
- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - Grayscale + Alpha + Extra: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp;
  - RGB + Alpha + Extra 8, 16 bpc;
  - CMYK: 32, 64 bpp;
  - CMYK + Extra: 8, 16 bpc;
  - Lab: 24, 48 bpp;
  - Lab + Extra: 8, 16 bpc;
  - HSL: 24, 48bpp;
  - HSL + Extra: 8, 16 bpc

- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - Grayscale + Alpha + Extra: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp;
  - RGB + Alpha + Extra 8, 16 bpc;
  - CMYK: 32, 64 bpp;
  - CMYK + Extra: 8, 16 bpc;
  - Lab: 24, 48 bpp;
  - Lab + Extra: 8, 16 bpc;
  - HSL: 24, 48 bpp;
  - HSL + Extra: 8, 16 bpc

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8 bpp;
  - Grayscale + Alpha: 16 bpp;
  - Grayscale + Alpha + Extra: 8 bpc;
  - RGB: 24 bpp;
  - RGB + Alpha: 32 bpp;
  - RGB + Alpha + Extra: 8 bpc;
  - CMYK: 32 bpp;
  - CMYK + Extra: 8 bpc;
  - Lab: 24 bpp;
  - Lab + Extra: 8 bpc
- IG_COMPRESSION_PACKED_BITS:
  - Indexed RGB: 1, 8 bpp;
  - Grayscale: 8 bpp;
  - Grayscale + Alpha: 16 bpp;
  - Grayscale + Alpha + Extra: 8 bpc;
  - RGB: 24 bpp;
  - RGB + Alpha: 32 bpp;
  - RGB + Alpha + Extra: 8 bpc;
  - CMYK: 32 bpp;
  - CMYK + Extra: 8 bpc;
  - Lab: 24 bpp;
  - Lab + Extra: 8 bpc

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
LOAD_FIRST_EXTRA_CHANNEL_AS_ALPHA	AT_BOOL	FALSE	TRUE, FALSE	This parameter specifies how ImageGear should load the first extra channel of a PSD image. If FALSE, ImageGear loads all extra channels according to their descriptors in the file header. If the channel contains transparency information, ImageGear loads it as Alpha channel. Otherwise ImageGear loads it as Extra channel. If TRUE, ImageGear loads first extra channel as Alpha channel. This mode provides backward compatibility with

				previous versions of ImageGear.
READ_LAYER_INDEX	INT	-1	Any integer, but no more than number of layers	When > -1 specifies a zero based index of layer mask image to load, otherwise reads the composition image.
READ_LAYER_MASK	AT_BOOL	FALSE	FALSE, TRUE	When TRUE PSD layer mask images are loaded as usual pages of multi-page file, otherwise layer masks are ignored.
SAVE_THUMBNAIL	AT_BOOL	FALSE	FALSE, TRUE	Gets/Sets thumbnail flag. If TRUE then thumbnail will be added to image.
THUMBNAIL_ENABLE	AT_BOOL	FALSE	FALSE, TRUE	When TRUE thumbnail reading function loads thumbnail image provided by PSD format.
THUMBNAIL_HEIGHT	UINT	64	Any positive value	Gets/Sets thumbnail height.
THUMBNAIL_WIDTH	UINT	64	Any positive value	Gets/Sets thumbnail width.

Comments:

The PSD file is considered by many in the computer graphics arts community as an industry standard.

The PSD is organized into 5 major segments of data: the header, 3 informational blocks, and the bitmap data. The short header always contains a "signature" of 8PPS, as well as these fields: width, height, and bit depth of the bitmap.

The first block of informational data is called the "Color Mode Data Block". It begins with a value for the length of the block. If the image has a palette, it is located here.

The next block is called the "Image Resources Block". Like the previous block, it first gives the length of the block. An ID field is filled with one of many possible values that indicate the structure where the data is stored. It may contain such parameters as resolution.

The last informational block is called the "Layer and Mask Instruction Block". After a value for the length of the block, it tells how many "layer records" follow. There is a record for each layer in the image. Each record begins with a channel ID and the length of the data in the record. After the records, a "Layer Mask" section may be stored, if applicable.

The bitmap data represents the last segment of a PSD file.

For historic reasons, ImageGear uses the IG_COMPRESSION_PACKED_BITS constant for saving Packed Bits compressed PSD images, and uses the IG_COMPRESSION_RLE constant to report Packed Bits compression when reading PSD images. For the PSD format, the terms "Packed Bits" and "RLE" are used synonymously.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats, 2d ed. Sebastopol, CA: O'Reilly & Associates, Inc., 1996.

1.2.6.7.5 AFX

Full Name	Auto FX Photographic Edges
Format ID	IG_FORMAT_AFX = 49
File Extension(s)	*.afx
Data Type	Raster Image
Data Encoding	JPEG Lossy
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

IG_COMPRESSION_JPEG - RGB: 24 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

Comments:

The Auto-FX file contains a small header followed by a JPEG datastream.

1.2.6.7.6 AVI

Full Name	MS Video for Windows video clip
Format ID	IG_FORMAT_AVI = 52
File Extension(s)	*.avi
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
FILE_NAME	String	""	any string	Obsolete. Not currently used.
IMAGE_IS_KEY_FRAME	AT_BOOL	TRUE	TRUE, FALSE	The filter will set this to TRUE or FALSE when reading an image depending on whether or not the image is marked as a key frame in the AVI file.

Comments:

AVI files are a special case of RIFF files. RIFF is the Resource Interchange File Format. This is a general purpose format for exchanging multimedia data types that was defined by Microsoft and IBM. An AVI file ("audio/video interleave") typically contains video and optionally audio which is synchronized to the video.

ImageGear can read video frame images from uncompressed and RLE compressed AVI files using this AVI format filter.

1.2.6.7.7 BMP

Full Name	Microsoft Windows Bitmap
Format ID	IG_FORMAT_BMP = 2
File Extension(s)	*.bmp, *dib
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Only for OS/2 BMP
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64, Java

ImageGear Supported Versions:

- Windows Bitmap version 5 (created for Windows 98, Windows 2000)
- Windows Bitmap version 4 (created for Windows 95, Windows NT 4.0)
- Windows Bitmap version 3 (created for Windows 3.x)
- Windows Bitmap version 2 (created for Windows 2.x)
- OS/2 Bitmap version 2
- OS/2 Bitmap version 1

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 15, 16, 24 bpp;
  - RGB + Alpha: 32 bpp.
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 15, 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
UPSIDE_DOWN	AT_BOOL	FALSE	FALSE, TRUE	If TRUE then the images will be saved upside-down
TYPE	UINT	BMP_TYPE_BMI	BMP_TYPE_BMC, BMP_TYPE_BMI, BMP_TYPE_BMI2	Type of BMP, see BMP_TYPE_... constants
COMPRESSION	DWORD	BMP_COMP_RGB	BMP_COMP_RLE4, BMP_COMP_RLE8, BMP_COMP_RGB	BMP compression, see BMP_COMP_... constants

B16_GRAY_SCANNER	AT_BOOL	FALSE	FALSE, TRUE	Vidar 12-bit scanner options
B16_GRAY_SCANTYPE	UINT	0		Vidar 12-bit scanner options

Comments:

The BMP format for versions 2.x - 4.x contains two headers. All Windows bitmap files begin with the same first header. They proceed with a data structure containing image information (the Bitmap Information Header), and end with the actual image data. If there is a palette (1, 4, 8-bit images), it is located between the bitmap information and the bitmap image data.

The first header identifies the format as BMP, and stores the file size and the address of the image. Two additional fields, Reserved1 and Reserved2, are not used and are set to 0.

The second header, known as the "bitmap information header", varies across the versions of Windows bitmaps. The second header for all bitmaps from version 2.x to 5.x have in common the following basic set of information: size of the secondary header in bytes, height and width of the image in pixels, the number of bit planes, the number of bits per pixel, compression scheme (0 = uncompressed, 1 = 4-bit RLE compression, 2 = 8-bit RLE compression, 3 = bitfields encoding was used), size of image in bytes, horizontal and vertical resolution in pixels per meter, the number of colors in the image, and the minimum number of important colors.

If the image is 16 or 32-bits per pixel in resolution, the compression field equals 3, and following the header are values for RedMask, GreenMask and BlueMask, rather than a palette. If the file is 4.x, there are values for an alpha component, color space type, x and y coordinates of red, green or blue endpoints, and gamma values for red, green, and blue coordinate scale values. The "ColorsImportant" field accommodates hardware that supports fewer colors than are contained by the image palette. The most significant colors are determined by counting their frequency of appearance. A value of zero means that all the colors in the image are significant.

The palette, or color table, varies in size depending on the number of colors in the image. This value is stored in the "ColorsUsed" field of the Bitmap Information Header. In the BMP format v. 3, the palette's structure is in "RGBQUAD" format. See the section entitled "Palettes" in for more information. 24-bit images do not use a palette, but rather store the color information directly in the image data.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1996.

1.2.6.7.8 BTR

Full Name	Brooktrout
Format ID	IG_FORMAT_BRK = 3
File Extension(s)	*.brk, *.301,
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported Versions:

Version 1

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetect
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G32D - Indexed RGB: 1 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G32D - Indexed RGB: 1 bpp

When saving an image, the image must have horizontal resolution of 200 DPI and vertical resolution of 100 or 200 DPI.

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
COMPRESSION	WORD	BTR_COMP_G3	BTR_COMP_G3BTR_COMP_G3_2D	Compression for saving.

Comments:

A Brooktrout file consists of a CCITT Group 3 (G3) compressed file with a 128-byte header designed by Brooktrout Technology. The header fields include a constant that identifies the file as Brooktrout, a version number, the horizontal and vertical resolutions of the image in dots/mm, the number of bits per pixel, and the number of pixels per line.

1.2.6.7.9 CAL

Full Name	CALS Raster
Format ID	IG_FORMAT_CAL = 4
File Extension(s)	*.cal, *.ras, *.cals
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64, Java

ImageGear Supported Versions:

- Type II (Tiles made possible)
- Type I

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB : 1 bpp

ImageGear Write Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB : 1 bpp

ImageGear Filter Control Parameters:

None

Comments:

The CALS file format was created as a graphics format specification by The Department of Defense to standardize the data exchange of logistics support operations across the military branches and military contractors. It is mandatory for most military document-handling applications. It is no longer used solely by the military and its contractors; other government agencies and commercial businesses have also adopted this format, including the aerospace, commercial computer, and medical industries.

There are two types of CALS files: Type I, and a newer, significantly more complicated Type II. Type II supports the use of tiles. Sometimes it acts a repository for a group of Type I files. Whether or not a Type II contains Type I images determines what kind of compression scheme is used. Type I is supported by ImageGear.

Type I and Type II files begin with a header that has the same format and size. It includes information about the source and destination documents, as well as image characteristics data. The data storage units under the header are each 128 bytes in length, and are referred to as records. These are written with 7-bit ASCII characters, making it more "human-readable" than most file format headers.

The image data follows the header. In a Type II file, if the data is a series of Type I images, the images are encoded with CCITT Group 4 compression. If they are Type II files, the data may either be uncompressed or encoded with CCITT Group 4. In addition, Type II data may be stored in tiles, wherein some, all, or none of the tiles may be compressed.

The Type II file contains several more substructures than Type I. Between the header and the image data are three groups of document formatting data. Other data preceding each image (or images), are "layout information" and a "Tile Index", that contains the address of each tile stored for the image.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.10 CLP

Full Name	Windows Clipboard
Format ID	IG_FORMAT_CLP = 5
File Extension(s)	*.clp
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Filter Control Parameters:

None

Comments:

The CLP file format represents a subset of a file format called Pictor PC Paint. The use for PC Paint format is to display images created by the PC Paint application to IBM display hardware (CGA, EGA, VGA, etc.).

The header of a CLP file is fairly short and simple, containing the file size (in bytes), and the height, width, and address of the image. The image data may be compressed or uncompressed. If it is compressed, the header contains two additional fields to give the number of bits per pixel of the packed data, and the address of the beginning of the packed run.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.11 CUR

Full Name	Windows cursor
Format ID	IG_FORMAT_CUR = 96
File Extension(s)	*.cur
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	Single alpha channel for read/write (see Comments).
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

Windows 3.x

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed + Extra: 1+1, 4+1, 8+1 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_RLE:
  - Indexed + Extra: 4+1, 8+1 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed + Extra: 1+1, 4+1, 8+1 bpp;
  - RGB + Alpha: 32 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
READ_AS_RGBA	AT_BOOL	FALSE	TRUE, FALSE	If TRUE, ImageGear reads CUR format as RGBA (RGB with alpha channel). See Comments for more detail.

Comments:

Cursor files consist of a file header (that is repeated several times), info headers, and cursor data. Cursor data contains an XOR mask bitmap and a monochrome AND mask bitmap. Whenever Windows draws a cursor, the AND bitmap is applied to whatever is on the screen. After that, the XOR bitmap is applied.

READ_AS_RGBA control parameter determines how ImageGear reads the 1-bit AND masks. If READ_AS_RGBA is FALSE, ImageGear reads AND mask into "Extra" channel. This mode preserves unchanged pixel values from the file. However, in this mode ImageGear displays only the XOR mask and ignores AND mask (Extra channel) during display, i.e. display is not transparent. If READ_AS_RGBA is TRUE, ImageGear reads CUR files as 32 bpp RGB + Alpha. This allows transparent display.

Files must be 255x255 pixels or less.

References Used:

A Jorn Daub EDV-Beratung - Glashutter Weg 105 - D-22889 Tangstedt

fileformats@daubnet.com

1.2.6.7.12 CUT

Full Name	Dr. Halo
Format ID	IG_FORMAT_CUT = 7
File Extension(s)	*.cut, *.pal (for the separately stored palette)
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, .NET, .NET64, Unix, Unix64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_RLE - Indexed RGB: 8 bpp

ImageGear Write Support:

IG_COMPRESSION_RLE - Indexed RGB: 8 bpp

ImageGear Filter Control Parameters:

None

Comments:

The Dr. Halo file format is associated with the HALO Image File Format Library, the Dr. Halo III paint program, and other applications created by Media Cybernetics.

This format consists of two separate files, one with an extension of .CUT and the other with an extension of .PAL. The .CUT file contains the image data and the .PAL file contains the color palette.

The .CUT file begins with a simple header of just three data fields: width and height of the image data (pixels by scan lines), and a reserved field intended for use with any future expansions of the header. The image data is always RLE-encoded and follows the header.

The palette file (.PAL) contains a header, with information about the type of palette used, and the size and maximum values of the Red, Green, and Blue components. The palette can be hardware-specific, in which case it contains additional data.

When a CUT image is loaded into ImageGear, the palette is initialized to a grayscale ramp. In order to achieve the original colors of the palette (PAL file), it must be loaded separately into the HIGEAR image.

When saving an image into the Dr. Halo format, ImageGear creates a .CUT file. In order to save the palette (PAL file), it must be saved separately.

See the section RLE for more information.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.13 DCX

Full Name	Paintbrush (Intel multi-page FAX format)
Format ID	IG_FORMAT_DCX = 8
File Extension(s)	*.dcx
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEWRITEPSUPPORT - multi-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
SAVE_COMPRESSED	AT_BOOL	TRUE	FALSE, TRUE	This parameters specify either compress output image or not. TRUE value cause to RLE compress image. FALSE cause to write image uncompressed.
ADD_IMAGE	AT_BOOL	TRUE	FALSE, TRUE	If this parameter is TRUE then image is added as additional page of multi-page image. If FALSE then the new image with single page is written

Comments:

This file format was designed to allow multiple PCX files to be stored in one file. This is especially desirable for multi-page faxes (for which the PCX format is often used). Up to 1024 PCX images can be stored in one DCX file.

The DCX construct begins with a simple header, then the PCX files are simply stored end-to-end, complete with their individual headers and palettes. An array in the header called "Pagetable" contains offsets to each PCX. The one piece of vital information not stored with the PCX files under a DCX is their original filenames. See the PCX section of this manual for more about PCX files.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.14 DICOM

Full Name	DICOM (Digital Imaging & Communication in Medicine)
Format ID	IG_FORMAT_DCM = 48
File Extension(s)	*.dicm, *.dcm
Data Type	Raster or vector image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix (Linux), Mac, .NET, .NET64

To support the DICOM format, attach the ImageGear Medical Component to Core ImageGear.

ImageGear Supported Versions:

- DICOM 3.0, 1991 - 2006

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEDELETESUPPORT - page deleting from multi-page file (only deletion of last page is supported)
- IG_FLTR_MPAGEWRITEPSUPPORT - multi-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 1, 8, 9-16, 32 bpp;
  - Indexed RGB: 8 bpp, 16 bpp (converted to 24-bit RGB during loading);
  - RGB: 24 bpp
  - Vector/waveform data
- IG_COMPRESSION_JPEG (lossy):
  - Grayscale: 8, 9-12 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (lossless):
  - Grayscale: 8, 9-16 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG2K:
  - Grayscale: 8, 9-16 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Grayscale: 8, 9-16 bpp;
  - Indexed RGB: 8 bpp, 16 bpp (converted to 24-bit RGB during loading);
  - RGB: 24 bpp

ImageGear also supports reading of Adobe PDF documents, encapsulated in DICOM files. See Adobe PDF format description for information on supported Adobe PDF features.

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8, 9-16, 32 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (lossy - baseline / process 1):
  - Grayscale: 8, 9-12 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (lossy - extended / process 2&4):
  - Grayscale: 8, 9-12 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (lossless):
  - Grayscale: 8, 9-16 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG2K:
  - Grayscale: 8, 9-16 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Grayscale: 8, 9-16 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 24 bpp

To be able to load encapsulated Adobe PDF documents, attach the ImageGear PDF Component.

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
DETECT_CONTINUOUS_RLE	AT_BOOL	TRUE	TRUE, FALSE	This parameter specifies what to do with RLE

				compressed images where RLE runs across row boundaries. If it is set to TRUE, ImageGear tries to detect and load images where RLE runs across row boundaries. Otherwise, ImageGear truncates any row overruns and decodes each row separately. This parameter does not affect the loading of properly encoded images, where each row is encoded separately.
LOAD_APPLY_LUT_FOR_32G	AT_BOOL	TRUE	TRUE, FALSE	Set to TRUE to apply LUT to pixel data for 17-32 bit grayscale images. Set to FALSE to leave pixel data intact. By default ImageGear applies LUT to pixel data for 17-32 bit per pixel grayscale images. If it is not necessary, set this control parameter to FALSE.
LOAD_CONCAT_REPEATED_DE	AT_BOOL	FALSE	TRUE, FALSE	TRUE to concatenate repeated data elements into one data element during loading. Parameter LOAD_CONCAT_REPEATED_DE allows you to load incompliant DICOM images where some data elements are cut into several repeated data elements. Instead of one data element containing an array of values, the data set contains several data elements, with the same group/element numbers pair, containing portions of the array. Specifically, there are images with look-up tables and palettes stored in this way. Set LOAD_CONCAT_REPEATED_DE parameter to TRUE to concatenate repeated data elements into one data element during loading. Otherwise (default), ImageGear loads all of the repeated elements in the same way as they are located in the file. Note that ImageGear does not allow saving of repeated data elements. If a data set contains repeated data elements, ImageGear will only write first repeated element to the file.
LOAD_CONVERTTO8G	AT_BOOL	FALSE	TRUE, FALSE	This control parameter has been deprecated and will be removed from the public API in a future release. Use IG_image_channel_depths_change after image loading to change its channel depths. Convert 9-16 bit gray to 8 on load.
LOAD_DETECTSKIPDIMSE	AT_BOOL	FALSE	TRUE, FALSE	This option controls loading process if a DICOM image contains DIMSE commands. DIMSE commands are a type of Data Element, with a group number of "0000" that are almost always removed by the DICOM network protocol before a transmitted image is saved to a disk file. However, sometimes they are found in the file and in such case ImageGear doesn't automatically recognize the file as a DICOM image. This is done because the DIMSE Tags are very hard to differentiate from other file formats that ImageGear supports. However, if DCM_CONTROL_LOAD_DETECT_SKIP_DIMSE is set to TRUE then the auto format detection skips over the DIMSE Tags when it attempts to decide if the file is DICOM or not.
LOAD_MASKALPHACHANNEL	AT_BOOL	TRUE	TRUE, FALSE	This option controls what would have done if an Alpha Channel image has been stuffed into the upper unused bits of a 16-bit image (Bits Stored < 16). These extra bits can be masked off or loaded along with the actual pixel value. If they are not masked off, you may need to alter the 16x8 LUT in order to display the image appropriately. If set to TRUE, the extra bits (the Alpha Channel) are masked off; if set to FALSE, the extra bits will be loaded into the DIB with the rest of the pixel.
LOAD_PAGENUMBER	UINT	1	Any positive integer value	This control parameter has been deprecated and will be removed from the public API in a future release. Use IG_fltr_load_file to load specific page of an image. Page number to load.
LOAD_SAVE_PIXDATA_TAG	AT_BOOL	FALSE	TRUE, FALSE	Set to TRUE to allow loading/saving pixel data to/from DataSet rather than to/from ImageGear DIB. When LOAD_SAVE_PIXDATA_TAG is TRUE, ImageGear does not read pixel data into a DIB, but rather creates an empty DIB, so the image cannot be displayed. When reading compressed image, ImageGear Medical

				treats PixelData tag as a Sequence, and places actual binary data into Item tags. This corresponds to the structure of compressed PixelData in DICOM files. When using this parameter for writing, make sure that Transfer Syntax matches actual Transfer Syntax of the PixelData element.
LOAD_SYNTAX	INT	MED_DCM_TS_AUTODETECT	enumIGMedTS values	This parameter controls the types of DICOM files ImageGear attempts to detect. If the file that is being loaded does not fall into the category specified by this control parameter it will be ignored and a IGE_CANT_DETECT_FORMAT error will be returned. • MED_DCM_TS_AUTODETECT = 9998 - ImageGear makes its best to determine the format of the DICOM file. • MED_DCM_TS_PART_10 = 9997 - only files with Part 10 Header will be detected. The Transfer Syntax of the file will be determined from the header and used to load the remainder of the image file. If you specify any standard DICOM Transfer Syntax, such as MED_DCM_TS_IMPLICIT_VR_LE or MED_DCM_TS_JPEG_LOSSY, the Medical Component will only load files having this Transfer Syntax.
LOAD_USE_8x8_LUT	AT_BOOL	TRUE	TRUE, FALSE	Set to TRUE to use 8x8 display LUT. Otherwise, use image's palette (the mechanism that was used in ImageGear v15.0 and earlier). Parameter LOAD_USE_8x8_LUT specifies the mechanism for display contrast adjustments of 8-bit grayscale images. Set to TRUE (default) to use 8x8 display LUT. Otherwise, use image's palette (the mechanism that was used in ImageGear v15.0 and earlier).
LOAD_USE_AUTO_WL_FOR_8G	AT_BOOL	TRUE	TRUE, FALSE	Set to TRUE to use auto window/level for 8g images, if VOI LUT is not present. Parameter LOAD_USE_AUTO_WL_FOR_8G affects loading of 8-bit grayscale images that do not have a VOI LUT (either a LUT sequence or window center/width values). Set to TRUE to use auto window/level for these images. Set to FALSE to apply no window/levelling (set contrast range to 0...255).
SAVE_ASPART10	AT_BOOL	TRUE	TRUE, FALSE	This parameter controls whether Meta Information Header is saved with the file or not. TRUE will cause the Header to be saved.
SAVE_GROUPLENGTHS	AT_BOOL	TRUE	TRUE, FALSE	This parameter controls the usage of Group Length values in a DICOM file. ImageGear treats these Data Elements as either on or off. That is they either are all included in each Group of Data Elements through the saving process or they are all absent. The internal Data Set that is attached to the HIGEAR does not contain Group Length Data Elements. When a DICOM file is to be written to disk they are computed and inserted if this parameter is set to TRUE. A value of TRUE indicates that Group Length values will be saved; FALSE indicates that they will not be saved.
SAVE_JPGQUALITY	UINT	70	1 - 100	This control parameter has been deprecated and will be removed from the public API in a future release. Please use QUALITY control parameter of JPEG filter instead. JPEG Quality setting 1-100
SAVE_LARGEST	AT_BOOL	FALSE	TRUE, FALSE	Controls whether Largest Image Pixel Value (0028,0107) is updated by ImageGear. If the original image Data Set did not contain a Data Element for Largest Image Pixel Value (0028,0107) and you set SAVE_LARGEST = TRUE, ImageGear scans the image and determines a value for this DE. Largest Image Pixel Value is included in the Data Set of the DICOM image being saved and contains the ImageGear-determined value. The value of the DE from the original Data Set (if any) is ignored. If you set SAVE_LARGEST = FALSE, ImageGear does not determine this value for you, and the Data Set of the image being saved does not include the Largest Image Pixel Value Data Element. However, if the original Data Set did contain this DE, ImageGear preserves and includes it in the

				Data Set being saved
SAVE_PLANARCONFIG	INT	MED_DCM_PLANAR_PIXEL_BY_PIXEL	• MED_DCM_PLANAR_PIXEL_BY_PIXEL • MED_DCM_PLANAR_PLANE_BY_PLANE	This parameter controls how the pixels are saved: • MED_DCM_PLANAR_PIXEL_BY_PIXEL: in normal RGB order ("pixel by pixel" configuration). • MED_DCM_PLANAR_PLANE_BY_PLANE: in a planar configuration, meaning that all Red, Blue, and Green pixels are saved in separate planes.
SAVE_SMALLEST	AT_BOOL	FALSE	TRUE, FALSE	This parameter controls whether ImageGear updates the Smallest Image Pixel Value (0028,0106). If the original image Data Set did not contain a Data Element for Smallest Image Pixel Value (0028,0106) and you set SAVE_SMALLEST = TRUE ImageGear scans the image and determines a value for this DE. Smallest Image Pixel Value are included in the Data Set of the DICOM image being saved, and contains the ImageGear-determined value. The value of the DE from the original Data Set (if any) are ignored. If you set SAVE_SMALLEST = FALSE ImageGear does not determine this value for you, and the Data Set of the image being saved does not include the Smallest Image Pixel Value Data Element. However, if the original Data Set did contain this DE, ImageGear preserves and includes it in the Data Set being saved.
SAVE_SYNTAX	INT	MED_DCM_TS_DEFAULT	enumIGMedTS values	This parameter controls how a DICOM file is to be formatted when it is written to disk. It does not control the file being Part 10 or Raw (see bSaveAsPart10), but controls how the non-Group 2 Data Elements are formatted. It also specifies the compression that will be used for Pixel Data.

Comments:

DICOM is a public standard created to provide a flexible and expandable means for storing, sharing, and transporting digital medical images. Today DICOM is the standard for medical imaging throughout the world.

DICOM image (alternately called Data Set) contains an ordered collection of attributes referred to as "Data Elements" that are related to one or more images. Each Data Element (DE) describes a single attribute of the image, patient, or study. The images themselves are also stored in DEs.

Each DICOM Data Set is transported through the Network, and consequently, stored in a file, using one of the defined Transfer Syntaxes. The Transfer Syntax of the DICOM file indicates whether the file uses Big Endian or Little Endian byte order, whether the image data is compressed or uncompressed, and if the DICOM Data Set uses Explicit or Implicit Value Representation (VR).

The Data Element (DE) is made up of the following parts:

- Tag
- Value Representation (optional)
- Value Length
- Value

Tag field identifies the type of information that is contained in the Value field (where the actual data is stored). The DICOM Data Dictionary (Part 6 of the specification) defines all possible public Data Element Tags that may be used. DICOM also allows applications to define and use private Tags and thus define their own Data Elements. Value Representation (VR) specifies the format of the Data Element Value, such as UL (unsigned long), ST (Short Text) or PN (Person Name). If Implicit Transfer Syntax is used, the VR field is omitted. It can be obtained from the standard or private Data Dictionary. Value Length is the length (in bytes) of the Data field.

DICOM also allows embedding (nesting) Data Sets within Data Sets. Nested Data Sets are implemented using a "Sequence of Items" (SQ), which is a special type of Data Element.

A special group of tags at the beginning of the file allows application to recognize the file as a DICOM image file, and includes information needed to decode the file (Transfer Syntax), and serial numbers to help locate and keep track of each field. This header is referred to as "File Meta Information Header", or "Part 10 Header" (since it is defined in Part 10 of the DICOM standard). While the standard clearly states that all DICOM image files must include a Part 10 Header, in reality one finds that many do not. Instead, the average DICOM image file is a simple data stream capture of the data into a file. This is called a "Raw Data" DICOM image file. In order to read a Raw DICOM the Transfer Syntax must be guessed at using a Transfer Syntax detection algorithm.

The actual pixel data for a DICOM image is stored in a Data Element, just like any other DICOM information. Image parameters, such as dimensions, bit depth, photometric interpretation etc., are also stored in Data Elements. The Tag for Pixel Data is called "Pixel Data" and has the Tag Number 7FE0, 0010. The Data Field in this Data Element contains all pixels for the image. Depending on the image's Transfer Syntax, the Pixel Data can be compressed or uncompressed.

References Used

Digital Imaging and Communication in Medicine (DICOM). Published by: National Electrical Manufacturers Association: http://medical.nema.org/dicom.html.

1.2.6.7.15 EPS

Full Name	Encapsulated PostScript File
Format ID	IG_FORMAT_EPS = 10
File Extension(s)	*.eps (may or may not contain preview image), *.epi (contains preview image), *epsf
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix (full support with ImageGear PDF component only), Unix64, MAC, .NET

ImageGear Supported Versions:

- Version 3.0
- Version 2.0
- Version 1.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- Images with uncompressed TIFF preview.

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
SAVE_PREVIEW	AT_BOOL	FALSE	TRUE, FALSE	If TRUE then EPS preview will be saved
START	DWORD	0		Position of start.
FITTING_METHOD	UINT	IG_EPS_FIT_ACTUAL	IG_EPS_FIT_PAGE, IG_EPS_FIT_ACTUAL, IG_EPS_FIT_SET	Fitting method, see IG_EPS_FIT_... constants
PIXEL_TO_PIXEL	AT_BOOL	FALSE	TRUE, FALSE	

PAGE_WIDTH	AT_DIMENSION	8500		Page width (100ths of an inch)
PAGE_HEIGHT	AT_DIMENSION	11000		Page height (100ths of an inch)
MARGINS	AT_RECT	{250, 250, 250, 250}		Margins (100ths of an inch)
X_DPI	UINT	300		X resolution
Y_DPI	UINT	300		Y resolution
TEXT_ENCODING	UINT	IG_PDF_TEXTENC_ASCII_HEX	IG_PDF_TEXTENC_NONE, IG_PDF_TEXTENC_ASCII_85, IG_PDF_TEXTENC_ASCII_HEX	Text encoding method, see IG_PDF_TEXTENC_ constants

Comments:

"PostScript" refers to a widely-supported general-purpose computer language that encodes text and graphics files for sharing with the many different hardware devices that support it. The full name for this language is "PostScript Page Description Language" (PDL).

An Encapsulated PostScript file stores (encapsulates) graphical or photographic images from a larger PostScript file. ImageGear currently supports the reading of EPS image of any bit depth as long as the preview image is TIF, uncompressed. ImageGear currently supports the writing of monochrome (1-bit), grayscale (8-bit gray level), and color RGB (24-bit) EPS images only.

The EPS file format begins with a PostScript language header. The data herein identifies the format as EPS, and gives the image title, creator, creation date, size and position of the image. Each line begins with a percent sign (%), which is normally interpreted in the PostScript language as the beginning of a comment line. Within the context of the EPS header, it takes on a different meaning.

Following the header is a block of PostScript code, which accomplishes the actual creation of the image.

The format proceeds with the bitmap data, or "graphics screen representation".

The EPI version of the EPS format, the version supported by ImageGear, contains an abridged interpretation of the image that is appended to the end of the file. It is usually smaller, and contains a lower resolution. One of the benefits of a preview image is that an application does not need to be able to interpret PostScript in order to display the image. Preview images are created with one of four file format types: TIFF, WMF, and EPS.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.16 EXIF-JPEG

Full Name	Exchangeable image file format (EXIF-JPEG)
Format ID	IG_FORMAT_EXIF_JPEG = 71
File Extension(s)	*.jpg, *.xif
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	Read, Write
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Read/write
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported Versions:

- Version 1.0 (1996)
- Version 1.1 (1997)
- Version 2.0 (1998)
- Version 2.1 (1998)
- Version 2.2 (2002)

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_JPEG (lossy, progressive):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
  - RGB + Alpha: 32 bpp;
  - CMYK: 32 bpp
- IG_COMPRESSION_JPEG (lossless):
  - Grayscale: 8, 16 bpp;
  - RGB: 24 bpp;
  - RGB + Alpha: 32 bpp;
  - CMYK: 32 bpp

ImageGear Write Support:

- IG_COMPRESSION_JPEG (lossy):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
  - RGB + Alpha: 32 bpp1
  - CMYK: 32 bpp2
- IG_COMPRESSION_JPEG (lossless):
  - Grayscale: 8, 16 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_JPEG (progressive):
  - Grayscale: 8, 12 bpp;
  - RGB: 24 bpp;

1) RGBA saving is disabled by default. Set JPEG-JFIF control parameter SAVE_ALLOW_RGBA to TRUE to enable it.

2) CMYK saving is enabled by default. You can disable it by setting JPEG-JFIF control parameter SAVE_ALLOW_CMYK to FALSE.

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
FLASHPIX_READY	AT_BOOL	FALSE	TRUE, FALSE	If TRUE the image will be saved with 64 interoperability, allowing lossless conversion to FlashPix format.
SAVE_JFIF_SEGMENT	AT_BOOL	FALSE	TRUE, FALSE	If TRUE the JFIF segment will be saved with image
SAVE_THUMBNAIL	AT_BOOL	TRUE	TRUE, FALSE	If TRUE the thumbnail will be saved with image
THUMBNAIL_WIDTH	UINT	160	Any positive value	Gets/Sets thumbnail width. Actual dimensions of the saved thumbnail will be adjusted to fit into rectangle specified by the THUMBNAIL_WIDTH and THUMBNAIL_HEIGHT parameters, preserving the ratio of image width and height.
THUMBNAIL_HEIGHT	UINT	120	Any positive value	Gets/Sets thumbnail height. Actual dimensions of the saved thumbnail will be adjusted to fit into rectangle specified by the THUMBNAIL_WIDTH and THUMBNAIL_HEIGHT parameters, preserving the ratio of image width and height.
THUMBNAIL_COMPRESSED	AT_BOOL	TRUE	TRUE, FALSE	For JPEG compressed EXIF. If TRUE the thumbnail will be JPEG compressed
LOAD_SCALE_DENOM	UINT	1	1, 2, 4, 8	If this parameter is set to any other than default value, ImageGear loads reduced version of the image, width and height of which are scaled by 1/load_scale_denom. This mode can be used for image preview, especially for those images that do not have embedded thumbnails, or where embedded thumbnails are smaller than desired. This mode allows you to make loading process 2-4 times faster.* * LOAD_SCALE_DENOM parameter affects the following formats: JFIF-JPEG, EXIF-JPEG. It does not affect TIFF-JPEG and other formats containing JPEG stream. Only Lossy and Progressive compressed images are supported. It affects all image loading functions and all image info functions.

The following control parameters of JFIF-JPEG format filter also affect EXIF-JPEG:

- SAVE_ALLOW_CMYK
- SAVE_ALLOW_RGBA

Comments:

The EXIF file format is based on existing formats. There are two kinds of EXIF format: compressed and uncompressed. Compressed EXIF is recorded in JPEG format with EXIF header saved in APP1 and APP2 marker segments. The APP2 segment is used when recording FlashPix extensions.

Uncompressed EXIF is recorded in TIFF Rev. 6.0 formats with two pages - the first is the main image, the second is a thumbnail (if it present). The EXIF header data is stored in TIFF 6.0 format for both compressed and uncompressed EXIF and include EXIF information (that is necessary) and GPS information (that is optional). Information specific to the camera system and not defined in TIFF is stored in private tags registered for EXIF. ImageGear EXIF support allows you to retrieve this information and send it to an application level and vice versa.

The EXIF image file specification also specifies the method for recording thumbnails. The reason for using the TIFF Rev. 6.0 tag format in the compressed file APP1 segment is to facilitate exchange of attribute data between EXIF compressed and uncompressed files.

Although the standard only allows uncompressed and JPEG-compressed EXIF images, and RGB color space, there are EXIF images that use other compressions and color spaces. In a sense, these images can be considered as JPEGs or TIFFs with additional EXIF metadata. ImageGear supports reading of such non-standard images, as well as writing of EXIF images with some non-standard compressions and color spaces.

References Used

Digital Still Camera Image File Format Standard (Exif) Version 2.0, Nov 1997, Japan Electronic Industry Development Association.

Digital Still Camera Image File Format Standard (Exif) Version 2.1, June 1998, Japan Electronic Industry Development Association.

1.2.6.7.17 EXIF-TIFF

Full Name	Exchangeable image file format (EXIF-TIFF)
Format ID	IG_FORMAT_EXIF_TIFF = 74
File Extension(s)	*.tif, *.xif
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	Read, Write
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Read only
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported Versions:

- Version 1.0 (1996)
- Version 1.1 (1997)
- Version 2.0 (1998)
- Version 2.1 (1998)
- Version 2.2 (2002)

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 1, 4, 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp;
  - RGB + Alpha: 8, 12, 16 bpc;
  - RGB + Alpha + Extra: 8, 12, 16 bpc;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc.
- IG_COMPRESSION_PACKED_BITS:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 1, 4, 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp;
  - RGB + Alpha: 8, 12, 16 bpc;
  - RGB + Alpha + Extra: 8, 12, 16 bpc;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc.
- IG_COMPRESSION_LZW:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 1, 4, 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp;
  - RGB + Alpha: 8, 12, 16 bpc;

  - RGB + Alpha + Extra: 8, 12, 16 bpc;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc.
- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 1, 4, 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp;
  - RGB + Alpha: 8, 12, 16 bpc;
  - RGB + Alpha + Extra: 8, 12, 16 bpc;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc.
- IG_COMPRESSION_HUFFMAN:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_CCITT_G32D:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_JPEG (Lossy):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
- IG_COMPRESSION_JPEG (Lossless):
  - Grayscale: 8, 16 bpp;
  - RGB: 24 bpp;

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8, 12, 16 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp.
- IG_COMPRESSION_PACKED_BITS:
  - Grayscale: 8, 12, 16 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp.
- IG_COMPRESSION_LZW:
  - Grayscale: 8, 12, 16 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp.
- IG_COMPRESSION_DEFLATE:
  - Grayscale: 8, 12, 16 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 32, 48, 64 bpp.
- IG_COMPRESSION_JPEG (Lossy):
  - Grayscale: 8, 12 bpp;

- RGB: 24, 36 bpp;
- IG_COMPRESSION_JPEG (Lossless):
    - Grayscale: 8, 16 bpp;
    - RGB: 24 bpp;

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
SAVE_THUMBNAIL	AT_BOOL	TRUE	TRUE, FALSE	If TRUE the thumbnail will be saved with image.
THUMBNAIL_WIDTH	UINT	160	Any positive value	Gets/Sets thumbnail width. Actual dimensions of the saved thumbnail will be adjusted to fit into rectangle specified by the THUMBNAIL_WIDTH and THUMBNAIL_HEIGHT parameters, preserving the ratio of image width and height.
THUMBNAIL_HEIGHT	UINT	120	Any positive value	Gets/Sets thumbnail height. Actual dimensions of the saved thumbnail will be adjusted to fit into rectangle specified by the THUMBNAIL_WIDTH and THUMBNAIL_HEIGHT parameters, preserving the ratio of image width and height.

Comments

The EXIF file format is based on existing formats. There are two kind of EXIF format: compressed and uncompressed. Compressed EXIF is recorded in JPEG format with EXIF header saved in APP1 and APP2 marker segments. The APP2 segment is used when recording FlashPix extensions.

Uncompressed EXIF is recorded in TIFF Rev. 6.0 formats with two pages - the first is the main image, the second is a thumbnail (if it present). The EXIF header data is stored in TIFF 6.0 format for both compressed and uncompressed EXIF and include EXIF information (that is necessary) and GPS information (that is optional). Information specific to the camera system and not defined in TIFF is stored in private tags registered for EXIF. ImageGear EXIF support allows you to retrieve this information and send it to an application level and vice versa.

The EXIF image file specification also specifies the method for recording thumbnails. The reason for using the TIFF Rev. 6.0 tag format in the compressed file APP1 segment is to facilitate exchange of attribute data between EXIF compressed and uncompressed files.

Although the standard only allows uncompressed and JPEG-compressed EXIF images, and RGB color space, there are EXIF images that use other compressions and color spaces. In a sense, these images can be considered as JPEGs or TIFFs with additional EXIF metadata. ImageGear supports reading of such non-standard images, as well as writing of EXIF images with some non-standard compressions and color spaces.

According to specification, EXIF-TIFF is a single-page format. However, some applications produce "multipage EXIF-TIFF" files, which are actually multipage TIFF files with EXIF metadata attached to each page. If you need to read such files, consider disabling auto-detection of EXIF-TIFF format, using IG_fltr_detect_set. ImageGear will then read such files as TIFF, ignoring EXIF metadata.

References Used

Digital Still Camera Image File Format Standard (Exif) Version 2.0, Nov 1997, Japan Electronic Industry Development Association.

Digital Still Camera Image File Format Standard (Exif) Version 2.1, June 1998, Japan Electronic Industry Development Association.

1.2.6.7.18 GEM

Full Name	GEM Raster
Format ID	IG_FORMAT_GEM = 13
File Extension(s)	*.gem
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

IG_COMPRESSION_RLE - 1, 4, 8 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

Comments:

A deal with the creators of Ventura Publisher made this a significant format in the desktop publishing arena. It was also distributed by PC systems manufacturers and was the native bitmap format of the Atari ST system.

The structure of a GEM Raster image file begins with a fixed-length header and is followed by the bitmap data.

The data fields of the header include the version number (always 1), image width (in pixels), and image height (in scan lines). A field named "Headerlength" contains a value of either 8 or 9, where 9 indicates that an optional field appears at the end, called "BitImageFlag". This field is directly tied with a "NumberOfPlanes" field. If "NumberOfPlanes" is greater than 1, the BitImageFlag indicates whether the image is color or grayscale. (BitImageFlag = 0 = color, BitImageFlag = 1 = grayscale).

Two fields in the header that are a little unusual among bitmap formats are the width and height of the pixels (in microns), and a field called "PatternLength". PatternLength is the length of any pattern that will be decoded. It is used by the RLE compression scheme, which assigns one of four different kind of codes to store each of the four different types of repeat data in the image. The "pattern code" indicates that a pattern is available. The pattern and repeat count are decoded while reading the file.

For more on RLE compression, see the ImageGear Supported Compressions Reference section.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.19 GIF

Full Name	CompuServe Graphics Interchange Format (GIF)
Format ID	IG_FORMAT_GIF = 14
File Extension(s)	*.gif
Data Type	Raster image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

To support the GIF format, attach the ImageGear LZW Component to Core ImageGear.

ImageGear Supported Versions:

- Version 89a, 1989: added ability to store both text and graphics in same file.
- Version 87a, 1987: first version; still read by major applications supporting GIF.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_MPAGEWRITEPSUPPORT - multi-page file writing

ImageGear Read Support:

IG_COMPRESSION_LZW - Indexed RGB: 1, 4, 8 bpp

ImageGear Write Support:

IG_COMPRESSION_LZW - Indexed RGB: 1, 4, 8 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
VERSION	INT	89	87; 89	Sets version of GIF file to be saved.
INTERLACE	AT_BOOL	FALSE	TRUE, FALSE	Sets GIF interlace mode to be used on saving.

Comments

This is a very popular format for storing graphics images on Web pages. It is also supported by most applications that handle graphical image data.

Unisys, the owner of the LZW compression used on all GIF files, declared at the end of 1995 that it will charge a royalty fee to all developers wishing to use this compression scheme. (This drove many developers to search for a replacement file format. A new file format called PNG ("Ping") was created. The PNG format is supported by ImageGear; see the PNG section for more information.) However, worldwide GIF-related patents expired in 2004 and the format is once again free to use without the need to pay royalties to Unisys.

While the GIF format is designed to store multiple images, few GIF format viewers support this. For this reason, it is not advisable to store more than one image in a GIF file.

The GIF layout is fairly complex; it can include several categories of "blocks" under which subcategories of blocks may occur.

For both the 89a and 87a versions, the first three blocks are the header, the Logical Source Descriptor, and the

Global Color Table. The header simply identifies the file as a GIF and gives the version number. The Logical Source Descriptor is very similar to a header, and is sometimes stored within the header. It contains information about the display screen and color table.

GIFs contain two kinds of color tables: a "Global Color Table" and "Local Color Table." The Global Color Table is used as a table for the pixel values of all images contained within the GIF file. Optionally, the Local Color Table block that is provided for each image can contain data specific to an individual image.

In addition to a local color table, each image is associated with another block of data, the Local Image Descriptor, provides the size and location of the image, and data about its color table.

GIF89a is equipped with four new types of blocks called "Control Extensions". The most significant group of these are the "Graphics Control Extension Blocks" that enable the simultaneous storing and displaying of textual and graphical data, including multiple images, resulting in a "multimedia presentation". The functions provided by these blocks include setting the transparency or opacity of the images, restoring or deleting images, and overlaying captions (that are not part of the actual bitmap) on images.

GIF image data is always stored in LZW-compressed form. The data may also be interlaced. Interlacing helps the appearance of an image as it displays while being decompressed, so that it "fades in". In a non-interlaced file, the presentation of the image data begins with row 1 and works downward to the last row of data. This method of display does not allow a quick preview of the whole image. When the data is interlaced, the lines are saved and displayed out of sequence. Every fourth row is displayed first and then filled in with every remaining fourth row, until all of the lines are displayed. This allows the eye to perceive the basic subject of the whole image before it is completely displayed.

See Also:

GIF Non-image Data Structure

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.20 Group 3

Full Name	Group 3 (G3)
Format ID	IG_FORMAT_G3 = 11
File Extension(s)	*.raw
Data Type	Raw image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	Win32, Win64, Unix, Java, Unix64

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp

ImageGear Write Support:

IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
WIDTH	DIMENSION	0	Any positive value	Image width
HEIGHT	DIMENSION	0	Any positive value	Image height
FILL_ORDER	UINT	1		FillOrder = 2, for inverse bit order in byte
PACKED	AT_BOOL	FALSE	TRUE, FALSE	Currently not in use. Reserved for the future.
K_FACTOR	UINT	2		K - factor for 2D coding

Comments:

ImageGear does not try to automatically detect raw Group3 data. Use one of the raw file loading functions, or specify format explicitly. Please see the sections Loading Images and Saving Images.

1.2.6.7.21 Group 3 2D

Full Name	Group 3 2D
Format ID	IG_FORMAT_G32D = 53
File Extension(s)	*.raw
Data Type	Raw image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	Win32, Win64, Unix, Java, Unix64

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_CCITT_G32D - Indexed RGB: 1 bpp

ImageGear Write Support:

IG_COMPRESSION_CCITT_G32D - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
WIDTH	DIMENSION	0	Any positive value	Image width
HEIGHT	DIMENSION	0	Any positive value	Image height
FILL_ORDER	UINT	1		FillOrder = 2, for inverse bit order in byte
PACKED	AT_BOOL	FALSE	TRUE, FALSE	Currently not in use. Reserved for the future.
K_FACTOR	UINT	2		K - factor for 2D coding

Comments:

ImageGear does not try to automatically detect raw Group3 data. Use one of the raw file loading functions, or specify format explicitly. Please see the sections Loading Images and Saving Images.

1.2.6.7.22 Group 4

Full Name	Group 4
Format ID	IG_FORMAT_G4 = 12
File Extension(s)	*.raw
Data Type	Raw image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	Win32, Win64, Unix, Java, Unix64

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Write Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
WIDTH	DIMENSION	0	Any positive value	Image width
HEIGHT	DIMENSION	0	Any positive value	Image height
FILL_ORDER	UINT	1		FillOrder = 2, for inverse bit order in byte
PACKED	AT_BOOL	FALSE	TRUE, FALSE	Currently not in use. Reserved for the future.
K_FACTOR	UINT	2		K - factor for 2D coding

Comments

ImageGear does not try to automatically detect raw Group3 data. Use one of the raw file loading functions, or specify format explicitly. Please see the sections Loading Images and Saving Images.

1.2.6.7.23 IBM AFP

Full Name	IBM AFP
Format ID	IG_FORMAT_AFP = 106
File Extension(s)	*.afp
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_BW:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_GRAY:
  - Indexed RGB: 4 bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
STITCH_TILES	AT_BOOL	FALSE	TRUE, FALSE	Set to TRUE to enable automatic tile stitching during image loading.

Comments:

The AFP Document File is the file, coded in IBM's MO:DCA format (Mixed Object Document Content Architecture). All MO:DCA objects contained in the AFP Document File include the following:

- Document Structure Objects
- Resource Objects
  - Font Objects
  - Overlay Objects
  - Page Segment Objects
- Graphics Objects
  - IM Image Objects
  - Image Objects (IOCA)
  - Graphics Objects (GOCA)
- Text Objects
- Bar Code Objects
- Object Containers

"IM" raster image objects are not supported by ImageGear. "IM" objects are legacy objects from an older version of AFP/MO:DCA specification. They have been superseded with IOCA objects.

1.2.6.7.24 IBM IOCA

Full Name	IBM IOCA (Image Object Content Architecture)
Format ID	IG_FORMAT_ICA =16
File Extension(s)	*.ica, *.mod
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_BW:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_GRAY:
  - Indexed RGB: 4bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
STITCH_TILES	AT_BOOL	FALSE	TRUE, FALSE	Set to TRUE to enable automatic tile stitching

				during image loading.

Comments:

IOCA files are most often used for document storage. They are not so unusual from most file formats, except for their naming conventions, which tend to be IBM-specific.

The general structure of an IOCA image file includes a "beginning segment", an "end segment", a header component called "Image Data Parameters", a palette, and the actual image data. Most IOCA images (less than 24-bit) contain a palette. The elements of the bitmap image are referred to as Image Data Elements (IDEs), that are called pixels by most other formats. The "Object Content" refers to the combination of the header and the image data.

IBM uses fields in the header called "self-defining fields". They each contain a type code, the length of the parameter, and then the actual parameter data. They include information as resolution, size, encoding scheme, and bit depth.

There are many optional parameters. Subsets of IOCA parameters are referred to as "function sets" and define different flavors of the IOCA, one being the MO:DCA, also supported by ImageGear. Examples of optional parameters include a tiling parameter, if the image is tiled, and a Band Image parameter, which signifies that the image is saved in "bands" ("bit planes" in other formats).

References Used

Image Object Content Architecture Reference, 2d ed., copyright International Business Machines Corporation, August 1991.

1.2.6.7.25 IBM MO:DCA

Full Name	IBM MO:DCA (Mixed Object Document Content Architecture)
Format ID	IG_FORMAT_MOD = 26
File Extension(s)	*.mod, *.ica
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC, .NET, .NET64

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_MPAGEWRITEPSUPPORT - multi-page file writing
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_BW:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC_GRAY:
  - Indexed RGB: 4bpp
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_IBM_MMR:
  - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description

STITCH_TILES	AT_BOOL	FALSE	TRUE, FALSE	Set to TRUE to enable automatic tile stitching during image loading.

Comments:

The MO:DCA header allows the storage of multiple IOCA images in one file. The MO:DCA format is an IOCA "wrapper". It is considered by IBM to be a "data stream controlling environment" for a group of IOCA images.

There are many optional parameters. Subsets of IOCA parameters are referred to as "function sets" and define different flavors of the IOCA, for example, the MO:DCA. The MO:DCA incorporates function sets "10" and "11".

Please see the description of IBM IOCA for further description.

References Used

Image Object Content Architecture Reference, 2d ed., copyright International Business Machines Corporation, August 1991.

1.2.6.7.26 ICO

Full Name	ICO (Windows icon)
Format ID	IG_FORMAT_ICO = 17
File Extension(s)	*.ico
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	Single alpha channel for read/write (see Comments).
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Windows 3.1
- Windows NT/95

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed + Extra: 1+1, 4+1, 8+1 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_RLE:
  - Indexed + Extra: 4+1, 8+1 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed + Extra: 1+1, 4+1, 8+1 bpp;
  - RGB + Alpha: 32 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
READ_AS_RGBA	AT_BOOL	FALSE	TRUE, FALSE	If TRUE, ImageGear reads CUR format as RGBA (RGB with alpha channel).

Comments:

More than one representation of the icon bitmap is stored in order to offer a choice of icons; the version most compatible to the output device is used. Support for read-write transparency masks has been added to the ICO filter. Transparency masks are placed into an alpha channel when the image is created. These masks can subsequently be set and applied to the main image.

The structure of an ICO file consists of four data sections: the header, the Resource Descriptor, and two representations of the image data per each icon (the color bitmap and the 1-bit masking bitmap).

The header identifies the file as an ICO and stores the number of icon images that are stored in the file.

The Resource Descriptor stores the image width and height, the number of colors used, and the offset from the beginning of the file to the image data.

The 1-bit masking bitmap defines the transparent portion of the bitmap.

READ_AS_RGBA control parameter determines how ImageGear reads the 1-bit AND masks. If READ_AS_RGBA is FALSE, ImageGear reads AND mask into "Extra" channel. This mode preserves unchanged pixel values from the file. However, in this mode ImageGear displays only the XOR mask and ignores AND mask (Extra channel) during display, i.e. display is not transparent. If READ_AS_RGBA is TRUE, ImageGear reads CUR files as 32 bpp RGB + Alpha. This allows transparent display.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

1.2.6.7.27 IFF

Full Name	Interchange File Format
Format ID	IG_FORMAT_IFF = 18
File Extension(s)	*.iff
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 1, 1985

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

None

Comments:

This type of file format is referred to as a "wrapper" because it can include any type of data that is encoded in any manner.

The basic organization of the IFF file format utilizes storage structures known as "chunks". A chunk is a block of data that contains its own header (that identifies the chunk size and type). This makes it easy for an IFF viewer to identify chunks and to skip over the ones that are not necessary.

The Header Chunk contains 17 fields, including the size of the header chunk, the identification of the chunk as a header, the size and origin of the image, data encoding (yes or no), and aspect ratio.

A "CMG Chunk" may follow, containing data specific to Amiga display hardware.

A "CMAP Chunk" contains the RGB palette for the image.

The "Body Chunk" (also called the ILBM or "interleaved bitmap"), is the image data itself. It is stored in an "interleaved" format, by bit plane. Interleaving allows for data with different resolutions to be neatly stored together. The data may be uncompressed or compressed using an RLE scheme. See RLE section for more information.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.28 IMG

Full Name	Xerox 9700 graphic image (IMG)
Format ID	IG_FORMAT_XRX = 46
File Extension(s)	*.img
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

Version 1.0

Version 2.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE - Indexed RGB: 1, 8 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

Comments:

The IMG header contains the width, height and resolution of the image. Following the header is compressed data. The sample values of a binary image are compressed and then encoded into a sequence of bits. Compression is achieved by predicting a pixel value based on pixel values that have already been computed. For example, the predicted value of a pixel may be that of the corresponding pixel on the previous scan line. Up to fifteen different compression techniques are used, each designed to remove redundancy from a certain kind of image - text characters, line art, and halftones of various screen frequencies. The algorithm adapts to the properties of the image by selecting the technique that will perform the best.

References Used

Xerox System Integration Standard. Raster Encoding Standard. Xerox, XNSS 178506, June 1990.

1.2.6.7.29 IMR

Full Name	IMRS-Raster image
Format ID	IG_FORMAT_IMR = 59
File Extension(s)	*.ima
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

1.2.6.7.30 IMT

Full Name	IMNET (Medical Image Format)
Format ID	IG_FORMAT_IMT = 20
File Extension(s)	*.imt
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G3 - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

The width of an image being saved must be a multiple of 8.

ImageGear Filter Control Parameters:

None

1.2.6.7.31 JPEG

Full Name	JPEG File Interchange Format
Format ID	IG_FORMAT_JPG = 21
File Extension(s)	*.ipg, *jpeg
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	Read, Write
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Read/write
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC, Java

ImageGear Supported Versions:

- Version 1.01 - 1991
- Version 1.02 - Added ability for thumbnails to be color-mapped and JPEG compressed.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_JPEG (Lossy):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
  - CMYK: 32 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_JPEG (Lossless):
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - CMYK: 32 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_JPEG (Progressive JPEG):
  - Grayscale: 8, 12 bpp;
  - RGB: 24 bpp;
  - CMYK: 32 bpp;
  - RGB + Alpha: 32 bpp

ImageGear Write Support:

- IG_COMPRESSION_JPEG (Lossy):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
  - RGB + Alpha: 32 bpp1
  - CMYK: 32 bpp2
- IG_COMPRESSION_JPEG (Lossless):
  - Grayscale: 8, 16 bpp;
  - RGB: 24 bpp;
- IG_COMPRESSION_JPEG (Progressive JPEG):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp

1. RGBA saving is disabled by default. Set SAVE_ALLOW_RGBA control parameter to TRUE to enable it.
2. CMYK saving is enabled by default. You can disable it by setting SAVE_ALLOW_CMYK control parameter to FALSE.

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
CALLER_ID	Long	21		Internal parameter. It will be removed from public API in a

				future release.
DECIMATION_TYPE	WORD	IG_JPG_DCM_2x2_1x1_1x1	See the "JPEG Decimation" section below.	Get/Set decimation value. For all available values see the "JPEG Decimation" section below.
ENTROPY_OPTIMIZE	AT_BOOL	FALSE	TRUE, FALSE	Get/Set entropy optimization flag. See the "JPEG Entropy Optimization" section below.
KEEP_ALPHA	AT_BOOL	FALSE	TRUE, FALSE	This parameter gets/sets alpha channel flag. If TRUE then alpha channel will be saved with original image (if it is present).
LOAD_SAVE_DCT	AT_BOOL	FALSE	TRUE, FALSE	Internal parameter. It will be removed from public API in a future release.
LOAD_SCALE_DENOM	UINT	1	1, 2, 4, 8	If this parameter is set to any other than default value, ImageGear loads reduced version of the image, width and height of which are scaled by 1/load_scale_denom. This mode can be used for image preview, especially for those images that do not have embedded thumbnails, or where embedded thumbnails are smaller than desired. This mode allows you to make loading process 2-4 times faster.*
LOAD_SCANS	UINT	0		Gets/Sets number of scans to load from Progressive JPEG image. See the "Progressive JPEG Scans" section below for details.
OLD_LOSSLESS_READ	AT_BOOL	FALSE	TRUE, FALSE	
PREDICTOR	AT_MODE	1		Gets/Sets predictor for lossless JPEG. See the "Lossless JPEG Predictor Settings" section below for details.
QUALITY	INT	70	Possible value in range [1,100]	Gets/Sets quality value for Lossy and Progressive JPEG compression. See the "JPEG Quality" section below for details.
SAVE_ALLOW_CMYK	AT_BOOL	TRUE	TRUE, FALSE	Set to TRUE (default) to allow saving JPEG CMYK images. Set to FALSE to save CMYK

				images to JPEG as RGB.
SAVE_ALLOW_LOSSY12BPCSAVING	AT_BOOL	TRUE	TRUE, FALSE	Controls the saving to Lossy JPEG format. If TRUE, saving of 12 bits per channel Lossy JPEG is allowed. This is the maximum channel depth supported by Lossy JPEG format. If source image's depth is more than 8 bits per channel, it will be saved to a 12 bpc JPEG. For example, 48-bit RGB image will be saved to 36-bit JPEG. Note that many viewers do not support 12 bpc JPEG. If FALSE, all images are saved to 8 bpc JPEG, regardless of their channel depth.
SAVE_ALLOW_RGBA	AT_BOOL	FALSE	TRUE, FALSE	Set this parameter to TRUE to enable RGBA JPEG saving. Note, that there is no official standard for RGBA JPEG. This format is recognized as CMYK rather than RGBA by most third-party software.
SAVE_JFIF_IN_EXIF	AT_BOOL	FALSE	TRUE, FALSE	Internal parameter. It will be removed from public API in a future release.
SAVE_THUMBNAIL	AT_BOOL	FALSE	FALSE, TRUE	Gets/Sets thumbnail flag. If TRUE then thumbnail will be added to image. See the "JPEG Decimation" section below for details.
SAVE_TYPE	AT_MODE	IG_JPG_LOSSY	IG_JPG_LOSSY, IG_JPG_LOSSLESS, IG_JPG_PROGRESSIVE	Get/Set type for output JPEG format. See the "Loading and Saving JPEG-Compressed Images" section below for details.
SCAN_INFO	VOID|LP	{ { 0, 0, 7, 1, 0 }, { 1, 5, 7, 1, 1 }, { 1, 5, 7, 1, 2 }, { 1, 5, 7, 1, 3 }, { 0, 0, 0, 0, 0 }, { 6, 63,7, 1, 1 }, { 6, 63,7, 1, 2 }, { 6, 63,7, 1, 3 }, { 1, 63,0, 0, 1 }, { 1, 63,0, 0, 2 }, { 1, 63,0, 0, 3 },};		Gets/Sets scan configuration for Progressive JPEG format. See the "Progressive JPEG Scans" section below for details.
SCAN_INFO_COUNT	UINT	sizeof(scan_info)/sizeof(scan_info[0]);		Gets/Sets length of array for previous parameter. See the "Progressive JPEG Scans" section below for details.
THUMBNAIL_COMPRESSED	AT_BOOL	FALSE	FALSE, TRUE	If TRUE then thumbnail will be compressed.
THUMBNAIL_HEIGHT	UINT	16		Gets/Sets thumbnail height.
THUMBNAIL_WIDTH	UINT	16		Gets/Sets thumbnail

width.

Comments:

JPEG is normally associated with the JPEG compression scheme, but it is also implemented into the JFIF file format. This format was developed to store JPEG-encoded data, and to exchange it between applications or operating systems that are normally incompatible.

The JPEG compression scheme was developed by the Joint Photographic Experts Group (created by the joining of a subgroup of the International Standards Organizations, called PEG (Photographic Experts Group) and a subgroup of the CCITT). Their common goal was to produce a standard for the transmission of graphics image data over networks and through color facsimile systems.

The header of the JFIF contains the version number, the image dots per inch (DPI), or dots per centimeter, and an optional thumbnail (miniature) RGB representation of the main image. Version 1.02 handles thumbnails differently by storing them separately, rather than in the identification marker of the header.

The raw JPEG data is surrounded by two markers, an "SOI" (start of image) marker, and an "EOI" (end of image) marker. See the section entitled Compression Schemes for more about JPEG compression.

JFIF is considered a non-proprietary file format. Many proprietary file formats contain JPEG data, incorporating their own application-specific data structures. Other non-proprietary formats that use JPEG-encoded data: TIFF file format, version 6.0.

Loading and Saving JPEG-Compressed Images

ImageGear supports the reading and writing of three types of JPEG compression: baseline JPEG (Lossy), Progressive JPEG, and Lossless JPEG. When you load a JPEG-compressed file, ImageGear detects the type of JPEG compression and decompress the image automatically. But if you want to save an image with a JPEG compression scheme other than baseline JPEG, you must use the "SAVE_TYPE" parameter to specify the type of JPEG compression.

Use these constants for "SAVE_TYPE" control parameter:

- IG_JPG_LOSSY- Lossy JPEG compression.
- IG_JPG_LOSSLESS - Lossless JPEG compression
- IG_JPG_PROGRESSIVE - Progressive JPEG compression.

JPEG Decimation

This table lists all possible decimation values:

IG_JPG_DCM_1x1_1x1_1x1	IG_JPG_DCM_2x1_1x1_1x1
IG_JPG_DCM_1x2_1x1_1x1	IG_JPG_DCM_2x2_1x1_1x1
IG_JPG_DCM_2x2_2x1_2x1	IG_JPG_DCM_4x2_1x1_1x1
IG_JPG_DCM_2x4_1x1_1x1	IG_JPG_DCM_4x1_1x1_1x1
IG_JPG_DCM_1x4_1x1_1x1	IG_JPG_DCM_4x1_2x1_2x1
IG_JPG_DCM_1x4_1x2_1x2	IG_JPG_DCM_4x4_2x2_2x2

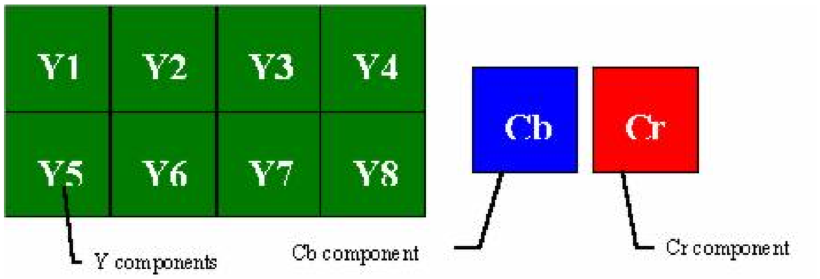
The format of these ImageGear decimation constants is:

```
IG_JPG_DCM_<H1>x<V1>_<H2>x<V2>_<H3>x<V3>,
```

where Hi, Vi = horizontal and vertical decimation values for the i-channel.

The following is a simple example of decimation. For a more detailed definition, please see the JPEG Specification.

A decimation setting of IG_JPG_DCM_4x2_1x1_1x1 would yield the following results:



As shown, 8 Y components in the source image have yielded one Cb and one Cr component. In general, this setting will reduce the quality of the compression, unless the image has many continuous tone areas.

- Maximum quality can be reached using a value of IG_JPG_DCM_1x1_1x1_1x1.
- Maximum compression ratio can be reached with a value of IG_JPG_DCM_4x4_2x2_2x2.
- The ImageGear default decimation value is: IG_JPG_DCM_2x2_1x1_1x1.

JPEG Thumbnails

The JPEG format can store thumbnails, which are small representations of the original image. These images are stored in uncompressed form and can significantly decrease your overall compression ratio. (Uncompressed thumbnails sometimes occupy more space than the original JPEG image when compressed). Use this option carefully.

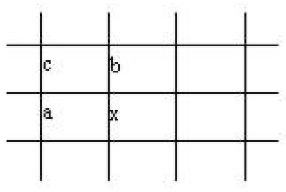
NOTE: The JPEG format does not allow the storage of "large" thumbnails. This is due to the marker segment length, which cannot be greater than 65,536 bytes. The maximum size of a color thumbnail is about 100x200 or 200x100 pixels, and the maximum size of a grayscale thumbnail is about 300x200 or 200x300 pixels.

Lossless JPEG Predictor Settings

The Lossless JPEG scheme is "predictive" in nature-it uses the values of surrounding pixels in addition to the value of the original pixel to calculate a predictor value, which it then subtracts from the value of the original pixel. The resulting pixel value will be reduced such that it can be compressed more than the original value. The higher the number of neighboring pixels used, the higher the compression will be.

Regardless of the predictor value setting, the quality of the image will remain the same. The difference is that if you choose to optimize for space by setting a high predictor value, you will have to give up some speed, as the decompression will take longer to perform.

ImageGear lets you set the predictor value using "PREDICTOR" control parameter. The ImageGear default for this setting is 1. The allowed range is 1-7. The graphic below shows a predictor (x) and three reconstructed samples (a,b,c) immediately to the left, immediately above, and diagonally to the left of the current sample



Lossless JPEG does not apply DCT for an image as per the Lossy JPEG compression. Instead, it uses a DPCM difference coding, which can be carried out with any one of seven different prediction modes. Correspondingly, the IG_CONTROL_JPG_PREDICTOR control parameter can be set to a value between 0 and 7. In Table 4, you will see what algorithm your setting of 1-7 will use, where Pr Px is the predictor, and Ra, Rb, and Rc are the reconstructed samples:

Value	**JPEG DIS Prediction**
0	no prediction
1	Px = Ra
2	Px = Rb
3	Px = Rc
4	Px = Ra + Rb - Rc
5	Px = Ra + ((Rb-Rc)/2)
6	Px = Rb + ((Ra-Rc)/2)
7	Px = (Ra + Rb)/2

JPEG Quality

The baseline JPEG specification calls for a quality setting. The lower the setting, the greater the number of original pixels lost, and therefore the smaller the resulting compressed file will be. ImageGear lets you set the quality of compression with values of type INT between 1-100, where 100 provides the highest retention of original pixel values. A setting of 100 does not mean that the image includes 100% of all original pixel values. With Lossy JPEG, there is no such thing as "no loss". Control parameter for setting Lossy JPEG quality is "QUALITY".

JPEG Entropy Optimization

Entropy optimization is relevant only to standard Lossy JPEG compression. If you set the parameter to TRUE, the default Huffman tables are not used. Instead, optimal Huffman tables are created for each component. This can bring a higher compression ratio but it takes more time for the compression.

Progressive JPEG Scans

A Progressive JPEG file stores more than just a copy of an image but rather several scans, each of which progressively adds a higher level of quality. Each scan contains a portion of the original image data. The purpose of this is to allow a very quick display of an image, beginning with a low-quality rendering and then increasing in quality as the remaining scans are added to it.

A Progressive JPEG image is stored as sequence of Huffman compressed blocks or "scans". Each scan contains the sequence of DCT coefficients in the given range. However, the coefficients are not complete. Only some of their bits will be stored in each scan.

ImageGear defines the following structure for holding the necessary configuration to write a JPEG image:

```
typedef struct tag AT_PJPEG_SCANINFO{
                LONG            Ss;
                LONG            Se;
                LONG            HBit;
                LONG            LBit;
                LONG            ChannelID;
}               AT_PJPEG_SCANINFO;
typedef AT_PJPEG_SCANINFO FAR* LPAT_PJPEG_SCANINFO;
```

The Ss and Se members of the AT_PJPEG_SCANINFO structure are used for spectral selection control coefficients:

```
LONG                    Ss;
LONG                       Se; - after applying DCT of 8x8 pixels we get 64
```

In Progressive coding, these coefficients are separated into different scans. Values Ss and Se specify the first and last number of the DCT coefficients that must be included in a given scan. The possible values are Ss = Se = 0 or 1<= Ss<=Se<=63. Please note following restrictions:

- The first coefficient (DC) cannot be encoded with the other coefficient (AC) in the single scan. In other words, the DC and AC coefficients cannot be in the same scan.
- Only scans that code DC coefficients may include interleaved blocks from more than one component. All other scans shall have only one component. For each component, a first DC scan shall precede any AC scans.

The HBit and LBit members of the AT_PJPEG_SCANINFO structure are used for successive approximation control:

```
        LONG HBit;
        LONG LBit;
```

If successive approximation is used, the DCT coefficients are reduced in precision by the point transform defined in the scan header. This is equivalent to taking some binary digits from each coefficient. HBit and LBit specify the high and low range of bits to take. For example, if HBit = 7 and LBit = 2, the scan will have the following original bits of the original DCT coefficient: 7,6,5,4,3,2.

The ChannelID member of the AT_PJPEG_SCANINFO structure is used to specify the number of components that will be encoded:

```
LONG ChannelID;
```

It can be set to one of the following values:

- interleaved scan which will only have DC coefficients of all components. This setting can only be used if Ss=Se=0.
- takes coefficients of first component.
- takes coefficients of second component.
- takes coefficients of third component.

There are two control parameters that operate with the AT_PJPEG_SCANINFO structure:

SCAN_INFO	Points to array of AT_PJPEG_SCANINFO elements.
SCAN_INFO_COUNT	Specifies the number of elements in this array. The nth entry of the SCAN_INFO array defines the configuration for the nth scan of the Progressive image.

The LOAD_SCANS control parameter specifies how many scans should be loaded. For example, if it is set to 1, the JPEG filter will load only the first scan of the image.

The following text blocks represent the different AT_PJPEG_SCANINFO structures that would be generated when loading a JPEG file using the ImageGear default settings for Progressive scans:

scan_config[0].Ss = 0;	scan_config[1].Ss = 1;
scan_config[0].Se = 0;	scan_config[1].Se = 5;
scan_config[0].HBit = 7;	scan_config[1].HBit = 7;
scan_config[0].LBit = 2;	scan_config[1].LBit = 2;
scan_config[0].ChannelID = 0;	scan_config[1].ChannelID = 1;
scan_config[2].Ss = 1;	scan_config[3].Ss = 1;
scan_config[2].Se = 5;	scan_config[3].Se = 5;
scan_config[2].HBit = 7;	scan_config[3].HBit = 7;
scan_config[2].LBit = 2;	scan_config[3].LBit = 2;
scan_config[2].ChannelID = 2;	scan_config[3].ChannelID = 3;
scan_config[4].Ss = 0;	scan_config[5].Ss = 1;

scan_config[4].Se = 0;	scan_config[5].Se = 5;
scan_config[4].HBit = 1;	scan_config[5].HBit = 1;
scan_config[4].LBit = 1;	scan_config[5].LBit = 1;
scan_config[4].ChannelID = 0;	scan_config[5].ChannelID = 1;
scan_config[6].Ss = 1;	scan_config[7].Ss = 1;
scan_config[6].Se = 5;	scan_config[7].Se = 5;
scan_config[6].HBit = 1;	scan_config[7].HBit = 1;
scan_config[6].LBit = 1;	scan_config[7].LBit = 1;
scan_config[6].ChannelID = 2;	scan_config[7].ChannelID = 3;
scan_config[8].Ss = 6;	scan_config[9].Ss = 6;
scan_config[8].Se = 63;	scan_config[9].Se = 63;
scan_config[8].HBit = 7;	scan_config[9].HBit = 7;
scan_config[8].LBit = 1;	scan_config[9].LBit = 1;
scan_config[8].ChannelID = 1;	scan_config[9].ChannelID = 2;
scan_config[10].Ss = 6;	scan_config[11].Ss = 0;
scan_config[10].Se = 63;	scan_config[11].Se = 0;
scan_config[10].HBit = 7;	scan_config[11].HBit = 0;
scan_config[10].LBit = 1;	scan_config[11].LBit = 0;
scan_config[10].ChannelID = 3;	scan_config[11].ChannelID = 0;
scan_config[12].Ss = 1;	scan_config[13].Ss = 1;
scan_config[12].Se = 63;	scan_config[13].Se = 63;
scan_config[12].HBit = 0;	scan_config[13].HBit = 0;
scan_config[12].LBit = 0;	scan_config[13].LBit = 0;
scan_config[12].ChannelID = 1;	scan_config[13].ChannelID = 2;
scan_config[14].Ss = 1;	
scan_config[14].Se = 63;	
scan_config[14].HBit = 0;	
scan_config[14].LBit = 0;	
scan_config[14].ChannelID = 3;	

References Used:

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.32 KFX

Full Name	KFX (Kofax Group4 image)
Format ID	IG_FORMAT_KFX = 22
File Extension(s)	*.kfx
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 3

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G4 - 1bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

1.2.6.7.33 LV

Full Name	LV (Lazer View format)
Format ID	IG_FORMAT_LV = 23
File Extension(s)	*.lv
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G3 - 1bpp
- IG_COMPRESSION_CCITT_G4 - 1bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

1.2.6.7.34 MAC

Full Name	MAC (Macintosh Paint)
Format ID	IG_FORMAT_MAC = 24
File Extension(s)	*.mac
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, MAC, UNIX

ImageGear Supported Versions:

Version 2.0 1989

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_RLE - 1 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

Comments:

Originally developed to store MacPaint graphics files, this format is now supported by many Macintosh applications. It is also exportable to the PC platform. It is always monochrome, and always has a fixed size of 576 pixels by 720 lines. The data, when uncompressed, is always 51,840 bytes in size.

Because this is a Macintosh format, it is organized as "forked" data. Each file consists of two forks, a "resource fork" and a "data fork". There is no code associated with this graphics format. The resource fork is always empty, and is easily merged together with the data fork when the file is exported to a PC platform.

The MacPaint data begins with a version number. If set to a value of 2, it indicates that paint patterns appear as the next structure. There are 38 possible patterns. These are generally not used, unless the file is being exported from one paint program to another.

The bitmap data begins at an offset of 512 bytes from the beginning of the file. The data is always compressed using the "PackBits" RLE compression scheme. The compressed data is stored in variable-length strips. See the description of RLE compression in the ImageGear Supported Compressions Reference section.

It the MacPaint file has been exported to a PC platform it contains a structure called the MacBinary header. This helps in reconstructing the resource fork if the file is returned to the Macintosh environment. A field in the MacBinary header holds the size of the fork. Other information includes the position of the file in the window, the version of the MacBinary header (I or II), the time and data of creation, and a SecondHeadLength field intended for future expansion of the MacPaint format should it require a secondary header.

References Used

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.35 MAC PICT

Full Name	Mac PICT
Format ID	IG_FORMAT_PCT = 30
File Extension(s)	*.pct, *.pict
Data Type	Metafile (2D raster, 2D geometry)
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Supports single alpha channel for read only. Alpha channel have to be 8-bit image.
ImageGear Platforms Support	WIN32, WIN64, MAC, UNIX

ImageGear Supported Versions:

- PICT 2 (for Color QuickDraw version 2). Added color and support for additional QuickDraw functions.
- PICT 1 (for Color QuickDraw version 1). Monochrome only.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp and 24+8 bit alpha
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp and 24+8 bit alpha
- IG_COMPRESSION_JPEG:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp and 24+8 bit alpha

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp (for 1, 4, 8 bpp, the files are ONLY saved as uncompressed if the image is 64x64 pixels or smaller. Otherwise, the image is saved as RLE compressed).
  - RGB: 24 bpp (Always saved as uncompressed)

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
FORCE_VERSION	AT_BOOL	FALSE	TRUE, FALSE	TRUE means saving of PCT in version 1.

Comments:

The PICT is one of the most widely-supported graphics file formats for the Macintosh.

PICT files begin with a fixed length header containing application-specific data, followed by fields that store the image size and location. If it is a PICT2 file, an additional header follows that contains the original resolution data of the image.

The bitmap data in the PICT2 format is referred as a Pixmap, from older terminology where Pixmap meant a bitmap with color.

The PICT file uses "opcodes". These are similar to the fields found in most file formats, and are associated with data that describes different shapes, lines, fill patterns, etc.

The lengthy list of opcodes is followed by the bitmap or "pixmap" data that describes the image data's address and resolution. The color table follows the opcodes. Next, source and destination rectangles are defined by their top left and lower right coordinates.

The pixel data, stored in the "PixData" field, is the last data to appear in the file. Each value is an index to the color table. This data is represented by 1, 2, or 4 bits.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.36 MSP

Full Name	MSP (MS Paint)
Format ID	IG_FORMAT_MSP = 25
File Extension(s)	*.msp
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 2.0
- Version 1.0

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE: Indexed RGB: 1 bpp
- IG_COMPRESSION_RLE: Indexed RGB: 1 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

Comments:

This was a popular format for storing line drawing and clip art images created with Windows applications, but has recently begun to be replaced by the Windows bitmap format (also supported by ImageGear see BMP).

For versions earlier than 2.0, the format begins with a 32-byte header and immediately proceeds with the bitmap data. The header information includes: the version of the file, the size and aspect ratio of the bitmap, and the width and height and aspect ratio (in pixels) of the output device used to render the bitmapped image. The bitmap data is uncompressed.

Files made with Version 2.0 and later always use RLE compression. In these files a "scan-line map" follows the header and precedes the data. It gives offsets to each scan line for instances when a particular scan line needs to be examined. All previous lines can remain compressed, while the needed line is located.

See RLE Compression under the ImageGear Supported Compressions Reference section for more information.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.37 NCR

Full Name	NCR
Format ID	IG_FORMAT_NCR = 27
File Extension(s)	*.ncr
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp
- IG_COMPRESSION_NONE - Indexed RGB: 1, 4 bpp

ImageGear Write Support:

- IG_COMPRESSION_CCITT_G4 - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
COMRESSION	WORD	0	0, NCR_CCITT_G4	This parameter has been retired.

Comments:

NCR is the black and white image compression format.

1.2.6.7.38 PBM

Full Name	PBM (Portable Bitmap File Format)
Format ID	IG_FORMAT_PBM = 28
File Extension(s)	*.pbm
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

October 1991 - last release

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_RAW:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp

ImageGear Write Support:

- IG_COMPRESSION_RAW:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp

ImageGear Filter Control Parameters:

None

Comments:

This is useful for quick and easy transfer of monochrome bitmap images, although the encoding scheme is not efficient in terms of storage space. This format, as well as the PGM, PNM, and PPM formats, are at the core of a set of utility programs also written by Jef Poskanzer. Among other things, these formats serve as intermediary storage methods for the conversion of other file formats.

The Portable Bitmap File Format structure is very simple. It begins with a short ASCII header that contains the file type identifier (magic number), the width and height of the image, and perhaps a comment line identifying the filename. Following white space (usually a carriage return) is the bitmap data. The number of bits is equal to the width * height. A pixel value of 0 indicates white, and a value of 1 indicates black.

The magic number of the header can have one of two values: either P1 or P4. P1 indicates that the bitmap data are to be read as ASCII decimal values. P4 indicates that the bitmap data are stored as plain bytes. Because 8 pixel values (1 bit each) are stored in one byte, the file is 8 times smaller than in the ASCII decimal format. White spaces are permitted in the P1 format but not permitted in the P4 format.

NOTE: Note: ImageGear uses IG_FORMAT_PBM filter to handle the whole family of formats: PBM (1-bit), PGM (grayscale), PPM (truecolor) and PNM (collective name for all of above). When saving image as IG_FORMAT_PBM format, ImageGear chooses particular format (PBM, PGM, PNM or PPM) depending on image bit depth.

See Also:

PGM, PPM, PNM

References Used

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest /McGraw-Hill, 1995.

PBM Specification by Jef Poskanzer, copyright © 1989, 1991.

1.2.6.7.39 PCD

Full Name	PCD (Kodak Photo CD)
Format ID	IG_FORMAT_PCD = 29
File Extension(s)	*.pcd
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

All valid PCD files in 5 different resolutions:

- Page 1: 768x512
- Page 2: 384x256
- Page 3: 192x128
- Page 4: 1536x1024
- Page 5: 3072x2048

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE - RGB: 24 bpp
- IG_COMPRESSION_HUFFMAN - RGB: 24 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

Comments:

"Photo CD" is the informally adopted name for files created using the Photo CD-ROM-based storage and retrieval system created by Kodak. The images are digitized versions of photographic images. Using a Photo CD player, images can be viewed on television. Although intended for photographic images, the data source does not necessarily have to be film. Due to the large storage capacity of the CD medium, this format supports very large and/or intricate images.

Images and their associated information are stored in groups called "sessions". Originally, they were stored at the rate of one session per CD, but later versions allowed multiple sessions per disc.

For each stored image, there are up to 5 bitmaps, each representing the image at a different resolution. The bitmaps at the lowest resolution are intended for such purposes as displaying thumbnails and previewing an image.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.40 PCX

Full Name	PCX (PC Paintbrush File Format)
Format ID	IG_FORMAT_PCX = 31
File Extension(s)	*.pcx
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 5.0 1991, Supports 24-bit RGB
- Version 3.0
- Version 2.8 Included color palette
- Version 2.5

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
SAVE_COMPRESSED	AT_BOOL	TRUE	FALSE, TRUE	If TRUE - image is saved with RLE Compression

Comments:

PCX is one of the oldest PC-based bitmap formats. It became well-known when Microsoft used it for their Paintbrush for Windows application and distributed it with every copy of Windows. This format is popular for fax documents because it allows them to be viewed within many popular paint and image display programs. If multiple images are desired in a PCX format, the format known as DCX, designed for this purpose, (and supported by ImageGear), may be used. Please see DCX file format for more information.

The main components of the PCX file format are the fixed length header, the image data, and if it is written for VGA display technology, the palette for the image (this appears as the last structure in the file). The header includes fields including the PCX version, the image size, resolution and position, and an encoding field that always has a value of 1; PCX data is always RLE. (For complex images, this may actually cause the bitmap data to increase in size).

The size and location of the palette associated with the image depends on the version of the PCX. When it was first developed, the limitations of the EGA card led to a palette that contained just 16 colors. PCX also supported CGA, so that the palette contained only 4 colors. Both of these palettes were stored in the palette array structure of the header. When the PCX was modified to display VGA, there was not enough room to store the palette in the header; subsequently, it was located at the end of the file.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.41 PGM

Full Name	PGM (Portable Graymap File Format)
Format ID	IG_FORMAT_PBM = 28 (see the Note below)
File Extension(s)	*.pgm
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear supports PGM file format via its IG_FORMAT_PBM format filter. Use IG_FORMAT_PBM to save grayscale images to PGM format.

ImageGear Supported Versions:

October 1991 - last release

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_RAW - Grayscale: 8, 16 bpp
- IG_COMPRESSION_ASCII - Grayscale: 8, 16 bpp

ImageGear Write Support:

- IG_COMPRESSION_RAW - Grayscale: 8, 16 bpp
- IG_COMPRESSION_ASCII - Grayscale: 8, 16 bpp

ImageGear Filter Control Parameters:

None

Comments:

This format quickly and easily transfers grayscale bitmap images. This format, as well as PBM, PNM, and PPM, are at the core of a set of utility programs also written by Jef Poskanzer. These formats serve as an intermediary storage methods for the conversion of other file formats.

The Portable Graymap File Format structure is very simple. It begins with a short ASCII header that contains the file type identifier (magic number), the width and height of the image, a "maximum gray value", and perhaps a comment line identifying the filename. The bitmap data follows white space (usually a carriage return). The number of pixels is equal to width * height. A pixel value of 0 indicates black, and a "maximum gray value" is equivalent to white.

The magic number of the header can have one of two values: either P2 or P5. P2 indicates that the bitmap data is read as ASCII decimal values. P5 indicates that the bitmap data is stored as plain bytes. This makes for a smaller and faster-to-read file.

If the maximum gray value does not exceed 255 (28 = 256 gray values from 0 to 255), each pixel is represented by a 8-bit sample. ImageGear loads such images as 8-bit grayscale. Otherwise, each pixel is represented by a 16-bit sample. ImageGear loads these images as 16-bit grayscale.

See Also:

PPM, PBM, PNM

References Used

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest /McGraw-Hill, 1995.

PGM Specification by Jef Poskanzer, copyright © 1989, 1991.

1.2.6.7.42 PNG

Full Name	PNG (Portable Network Graphics)
Format ID	IG_FORMAT_PNG = 33
File Extension(s)	*.png
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	Read, Write
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Supports single alpha channel for read/write.
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- 10th draft (future drafts will be backward-compatible)
- PNG v2

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 2, 4, 8 bpp;
  - Grayscale: 1, 2, 4, 8, 16 bpp;
  - Grayscale + Alpha: 8, 16 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp

ImageGear Write Support:

- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8, 16 bpp;
  - Grayscale + Alpha: 16, 32 bpp;
  - RGB: 24, 48 bpp;
  - RGB + Alpha: 32, 64 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
BUFFER_SIZE	DWORD	0		Size of the buffer for strip if STRIP_CONFIG is IG_PNG_STRIP_FIXED_BUFFER
COMP_LEVEL	UINT	IG_PNG_DEFAULT_COMPRESSION	IG_PNG_DEFAULT_COMPRESSION, IG_PNG_MIN_COMPRESSION, IG_PNG_MAX_COMPRESSION	Compression level. It must be between IG_PNG_MIN_COMPRESSION and IG_PNG_MAX_COMPRESSION. Better compression takes more time for compression.
KEEP_ALPHA	AT_BOOL	TRUE	TRUE, FALSE	When loading: if this parameter is TRUE, then alpha channel is loaded into Alpha DIB. Otherwise, the image is transformed into RGB, by compositing over the background supplied in the bKGD chunk, if it is present, or over the default background. When saving: if this parameter is TRUE, and alpha DIB is present, it will be written to the file.
SAVE_INDEXED_GRAY_AS_GRAY	AT_BOOL	FALSE	FALSE, TRUE	Affects saving of images that have grayscale palette. If TRUE then ImageGear saves the image as Grayscale (type 0). Otherwise, ImageGear saves the image as Paletted (type 3).
STRIP_CONFIG	INT	IG_PNG_STRIP_FIXED_COUNT = 0	IG_PNG_STRIP_FIXED_COUNT,IG_PNG_STRIP_FIXED_BUFFER	Format of PNG strip, see IG_PNG_STRIP_... constants
STRIP_COUNT	INT	0		Number of lines in one strip if STRIP_CONFIG is IG_PNG_STRIP_FIXED_COUNT

Comments:

The PNG (pronounced "Ping") format was created out of reaction to Unisys's announcement that it would begin requiring royalty fees for use of its LZW compression scheme. This was the compression scheme for the widely-used GIF format (found in Web pages and online library images).

Thomas Boutell and a host of other programmers began working to devise a new file format to eliminate the need for payment of royalty fees. The result was a file format that offers better compression than GIF, and adds features GIF doesn't offer, including truecolor, and full alpha channel and gamma correction.

The basic organization of the PNG file format utilizes storage structures known as "chunks". A chunk is a block of data that contains its own header (identifying the chunk size and type). This makes it easy for a PNG viewer to identify chunks and to skip over the ones that are not necessary.

The first entry in a PNG file is the "PNG signature" that identifies the format as PNG. The file then proceeds with a series of chunks.

The IHDR Image Header, or IHDR Chunk, contains a number of fields including the height, width, depth, color type, and compression type of the image. The only valid compression value is 0, which indicates the PNG's custom compression scheme, a deflate/inflate compression with a 32k sliding window. This is a derivative of ZZ77, the precursor to LZW. ZZ77 is the compression scheme used by pkzip software.

The "PLTE Palette Chunk" contains 1 to 256 palette entries. This is present if the type field of the header chunk is set to 3. The number of colors cannot exceed the range provided by the bit depth.

The image data is stored in "IDAT Image Data Chunks". These are subdivided into chunks whose size is usually determined by the size of the encoder's buffer. The data may be stored in "interlaced order" allowing the image to be "faded in". The data may also be interlaced. Interlacing helps the appearance of an image as it displays while being decompressed, so that it "fades in". In a non-interlaced file, the presentation of the image data begins with row 1 and works downward to the last row of data. This method of display does not allow a quick preview of the whole image. When the data is interlaced, the lines are saved and displayed out of sequence. Every fourth row is displayed first and then filled in with every remaining fourth row, until all of the lines are displayed. This allows the eye to perceive the basic subject of the whole image before it is completely displayed.

"Ancillary Chunks" are optional. They must appear before the first IDAT and after the PLTE. One chunk is the "bKGD chunk" that sets the default background color for the image. Two of the other ancillary chunks are the "hHist Chunk" or histogram chunk, and the "tEXt" chunk. The histogram chunk appears if there is a palette. It stores the frequency of each color of the palette as it occurs in the image data. If the application doesn't support all of the colors in the palette, the histogram can be used to choose a subset of colors. There may be any number of text chunks. They can vary in length from 0 to the maximum chunk size. They include the image author, copyright information, and any desired comments.

Questions about PNG can be e-mailed to: png-info@uunet.uu.net

References Used:

Murray, James D. "Graphic Image Format FAQ 3-4". James D. Murray, 1994-1996.

PNG (Portable Network Graphics), tenth draft. Page 5, copyright Thomas Boutell, May 1995.

Wegner, Tim. "Coding for PNG Graphics", "PC Techniques", Feb/Mar 1996, pp 32-38.

1.2.6.7.43 PNM

Full Name	PNM (Portable Any-Map File Format)
Format ID	IG_FORMAT_PBM = 28 (See the Note below)
File Extension(s)	*.pnm
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear supports PNM file format via its IG_FORMAT_PBM format filter. Use IG_FORMAT_PBM to save images to PNM format.

ImageGear Supported Versions:

October 1991 - last release

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_RAW:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp

ImageGear Write Support:

- IG_COMPRESSION_RAW:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII:
  - Indexed RGB: 1 bpp;
  - Grayscale: 8, 16 bpp;
  - RGB: 24, 48 bpp

ImageGear Filter Control Parameters:

None

Comments:

The PNM format is useful for quick and easy transfer of uncomplicated monochrome, grayscale, or color bitmap images. This format, as well as the PBM, PGM, and PPM formats, are at the core of a set of utility programs also written by Jef Poskanzer. These formats serve as intermediary storage methods for the conversion of other file formats. For example, a function called gifftoppm translates a GIF file to a PPM, from where it can be then translated to a TIFF using the pnmtotiff.

The Portable Anymap File Format structure is very simple. The "Anymap" portion of its name refers to the fact that it can be one of three types of UNIX bitmap file formats: the Portable Bitmap File Format (PBM), the Portable Graymap File Format (PGM), or the Portable Pixelmap File Format (PPM). A PNM begins with a short ASCII header that contains the file type identifier (magic number), the width and height of the image, a "maximum color-component value" if it is a PPM, a "maximum gray value" if it is a PGM, and perhaps a comment line identifying the filename. Following white space (usually a carriage return) is the bitmap data. For PGM and PPM files, the number of pixels is equal to width * height (whereas, in a PBM file, this calculation gives you the number of bits).

The magic number of the header can have one of six values depending on whether the bitmap is in PBM, PGM, or PPM format. Each format can have one of two magic numbers depending on whether the pixel data is stored in ASCII decimal or plain bytes. See the descriptions for the individual formats (these are all supported by ImageGear) for more details.

References Used

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest /McGraw-Hill, 1995.

PBM, PGM, PPM, PNM Specifications by Jef Poskanzer, copyright 1989, 1991.

1.2.6.7.44 PPM

Full Name	PPM (Portable Pixmap File Format)
Format ID	IG_FORMAT_PBM = 28 (See the Note below)
File Extension(s)	*.ppm
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear supports PPM file format via its IG_FORMAT_PBM format filter. Use IG_FORMAT_PBM to save truecolor images to PPM format.

ImageGear Supported Versions:

October 1991 - last release

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_RAW - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII - RGB: 24, 48 bpp

ImageGear Write Support:

- IG_COMPRESSION_RAW - RGB: 24, 48 bpp
- IG_COMPRESSION_ASCII - RGB: 24, 48 bpp

ImageGear Filter Control Parameters:

None

Comments:

The PPM format is useful for quick and easy transfer of color bitmap images. This format, as well as the PBM, PGM, and PNM formats, are at the core of a set of utility programs also written by Jef Poskanzer. These formats serve as intermediary storage methods for the conversion of other file formats. For example, a function called gifftoppm translates a GIF file to a PPM, where it can translate to a TIFF using the pnmtotiff.

The Portable Pixmap File Format structure is very simple. It begins with a short ASCII header that contains the file type identifier (magic number), the width and height of the image, a "maximum color-component value", and perhaps a comment line identifying the filename. Following white space (usually a carriage return) is the bitmap data. The number of pixels is equal to width * height, with each pixel being represented by three bytes: one each for Red, Green, and Blue color components, respectively.

The magic number of the header can have one of two values: either P3 or P6. P3 indicates that the bitmap data is read as ASCII decimal values. P6 indicates that the bitmap data is stored as plain bytes. This makes for a smaller and faster-to-read file.

If the maximum gray value exceeds 255 (28 = 256 gray values from 0 to 255), each pixel is represented by three 16-bit RGB samples, making a total of 48 bits per pixel. ImageGear loads such images to 24-bit RGB.

See Also:

PGM, PBM, PNM

References Used:

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest /McGraw-Hill, 1995.

PPM Specification by Jef Poskanzer, copyright © 1989, 1991.

1.2.6.7.45 QuickTime

Full Name	QT (Quick Time movie)
Format ID	IG_FORMAT_QUICKTIMEJPEG = 76
File Extension(s)	*.jpg, *.mov, *.qt
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

IG_COMPRESSION_JPEG - RGB: 24 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

None

Comments:

QuickTime format has been developed by Apple, inc. as a container for various media such as video and audio clips and still images.

ImageGear Core component supports loading of the first frame from JPEG- and Motion JPEG compressed QuickTime files.

1.2.6.7.46 RAS

Full Name	RAS (Sun Raster Data Format)
Format ID	IG_FORMAT_RAS = 37
File Extension(s)	*.ras
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 1, 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
SAVE_COMPRESSED	INT	0 (FALSE)	TRUE, FALSE	Compression flag. If TRUE the saving image will be RLE compressed.

Comments:

Bitmap images used under the SunOS system and UNIX imaging applications are usually stored in Sun Raster form.

The Sun Raster header contains image data (size and type), and the type and size of the colormap, if present. It also contains a Sun Raster identifying tag called "MagicNumber," which always contains the same value. A type field identifies the version of Sun Raster, the most common are called "Old" and "Standard". These are actually the same format and indicate that the image is not compressed. Other possible versions of Sun Raster files include TIFF and IFF, meaning that the image data was converted from one of these formats.

Following the header is a colormap, if applicable. Most 24 or 32-bit raster images do not use a colormap, but rather store the color values directly with the image data. This is known as "truecolor".

The last element of the Sun Raster file is the image data itself. It is usually in 2D raster format.

References Used:

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.47 RAW

Full Name	RAW (Raw image)
Format ID	IG_FORMAT_RAW = 58
File Extension(s)	*.raw
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Yes
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 9-16 bpp;
  - RGB: 24 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_LZW:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 9-16 bbp;
  - RGB: 24 bpp;
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_CCITT_G32D:
  - Indexed RGB: 1 bpp
- IG_COMPRESSION_ABIC:
  - Indexed RGB: 1 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
ALIGNMENT	UINT	0	0, 1, 2, 4	Row alignment. • 0: DWORD alignment • 1: BYTE alignment • 2: WORD alignment • 4: DWORD alignment
BITS_PER_PIXEL	UINT	0		Used internally

COMPRESSION	AT_MODE	0		Used internally
FILL_ORDER	INT	0		Used internally
HEIGHT	AT_DIMENSION	0		Used internally
UNCOMPRESSED_PACKED	AT_BOOL	FALSE	FALSE, TRUE	Set to TRUE to read uncompressed packed RAW format, where pixels are not padded to a byte boundary. Set to FALSE to read unpacked format. For example, two 12-bit pixels occupy 3 bytes in the packed format, and 4 bytes in the unpacked format.
WIDTH	AT_DIMENSION	0		Used internally

Comments:

A raw image file contains no header or identifying information. Also, ImageGear can load images of proprietary or unsupported formats as Raw data. Since ImageGear cannot obtain parameters, such as width, height or bits per pixel from the file, the application should specify them. For uncompressed images, ImageGear assumes BMP row order (bottom line goes first). If the RAW image uses the other row order, use IG_IP_flip function to flip it vertically after loading.

1.2.6.7.48 Scitex CT

Full Name	SciTex (Scitex CT)
Format ID	IG_FORMAT_SCI_CT = 91
File Extension(s)	*.sct
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE - CMYK: 32 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE - CMYK: 32 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
HEIGHT_IN_UNITS	DOUBLE			Height of the picture, in measurement units.
WIDTH_IN_UNITS	DOUBLE			Width of the picture, in measurement units.
FILE_TYPE	WORD	"CT"	"CT"	Scitex file type. The only "CT" - Continuous Tone Picture is currently supported.
UNITS_MEASUREMENT	BYTE		0 1	Units of measurement (0 mm, 1 inches)
SCAN_DIRECTION	BYTE			ScanDirection is a bitmap indicating the position of the image source on the scanner. The bitfields are defined as follows: Bit 0 0 = Top to bottom, 1 = Bottom to top Bit 1 0 = Left to right, 1 = Right to left Bit 2 0 = No rotation, 1 = 90 degree counter-clockwise rotation Bits 3:7 Undefined (always 0)

Comments:

ImageGear has read and write support for Scitex HandShake Continuous Tone Picture files. This is the native format used by Scitex scanners and printers for high-end image processing and color separation.

Scitex CT files store uncompressed, CMYK true-color raster data. They contain a Control Block, a Parameters Block, and the image data. Scitex CT images are typically four-color separation, CMYK, line-interleaved raster data. The separations are always stored by scan line and in the order C-M-Y-K (cyan-magenta-yellow-black). A color pixel value have up to 16 separations (128 bits) in size. Separations 1 through 4 are defined in order (C-M-Y-K). Separations 5 through 16 are reserved for future expansion of the format, as shown below.

Each row or Scitex CT image data is stored in separated color. The first separation's row data is followed by the second, and so forth, up to the number of separations specified by NumColorSeparations. Only the data for the separations defined by the SeparationsBitMask field is actually stored in the CT file. Each pixel can contain up to 16 separation components and each component is one byte in size. A CMYK pixel contains four components and is of a 32-bit size. Remember that the data is not stored by pixel, but by separation. If rows contain odd numbers of bytes, the zero padding byte will be added to the end of each separation to preserve word alignment.

You can set ImageGear CMYK Support level to IG_CONVERT_TO_RGB to convert CMYK images to 24-bit RGB during loading. However, use of this mode has been deprecated and will be removed from the public API in a future release. We recommend to load the image in its native format, and then convert to desired color space if needed.

References Used

Copyright 1994, 1996, O'Reilly & Associates, Inc.

1.2.6.7.49 SGI

Full Name	SGI (Silicon Graphics Image)
Format ID	IG_FORMAT_SGI = 38
File Extension(s)	*.sgi, *.bw, *.rgb, *.rgba
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Supports single alpha channel for read and write.
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_RLE:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp;
  - RGB + Alpha: 32 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp;
  - RGB + Alpha: 32 bpp
- IG_COMPRESSION_RLE:
  - Grayscale: 8 bbp;
  - RGB: 24 bpp;
  - Grayscale + Alpha: 16 bpp;
  - RGB + Alpha: 32 bpp;

ImageGear Filter Control Parameters:

None

Comments:

This format was developed for use with the SGI image library included on most Silicon Graphics computers. Most SGI images are black and white.

The major components of an SGI file are the 512-byte header, a "scan-line offset table", and the bitmap header. SGI is one of the few formats to use a scan-line offset table.

The fields of the header structure include a compression flag (1 = compressed), the height and width (in pixels) of the image, the number of bit planes, the highest pixel value and the lowest pixel value, the name of the image, and pixel format. Pixel format can indicate the number of color channels, whether the image is dithered to a single channel, and whether the bitmap image is actually a color map for other images.

The bitmap data is stored up-side-down-the first scan line is at the bottom of the bitmap. If the data is RLE, a scan-line offset table is present, following the header and preceding the bitmap. This increases compression further by allowing repeated offsets to the same scan line, if several scan lines have the same value. A grayscale image may even refer to the same scan line from three different bit fields. For this reason, and because SGI data that is compressed can be stored in any scan-line order, the offset table must not be ignored.

References Used

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats, 2d ed. Sebastopol, CA: O'Reilly & Associates, Inc., 1996.

1.2.6.7.50 TGA

Full Name	TGA (Truevision Targa)
Format ID	IG_FORMAT_TGA = 39
File Extension(s)	*.tga, *.tpic
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	Single alpha channel for read/write
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 2.0, 1991
- Version 1.0, 1984

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Grayscale: 1 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 15, 24 bpp;
  - RGB 15 bpp + Alpha 1 bpp;
  - RGB 24 bpp + Alpha 8 bpp;
  - RGB 15 bpp + Premultiplied Alpha 1 bpp;
  - RGB 24 bpp + Premultiplied Alpha 8 bpp;
  - RGB 15 bpp + Extra 1 bpp;
  - RGB 24 bpp + Extra 8 bpp;
- IG_COMPRESSION_RLE:
  - Grayscale: 1 bpp;
  - Indexed RGB: 8 bpp;
  - RGB: 15, 24 bpp;
  - RGB 15 bpp + Alpha 1 bpp;
  - RGB 24 bpp + Alpha 8 bpp;
  - RGB 15 bpp + Premultiplied Alpha 1 bpp;
  - RGB 24 bpp + Premultiplied Alpha 8 bpp;
  - RGB 15 bpp + Extra 1 bpp;
  - RGB 24 bpp + Extra 8 bpp;

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 8 bpp;
  - RGB: 15, 24 bpp;
  - RGB 15 bpp + Alpha 1 bpp;
  - RGB 24 bpp + Alpha 8 bpp;
  - RGB 15 bpp + Extra 1 bpp;
  - RGB 24 bpp + Extra 8 bpp;
- IG_COMPRESSION_RLE:

- Indexed RGB: 8 bpp;
- RGB: 15, 24 bpp;
- RGB 15 bpp + Alpha 1 bpp;
- RGB 24 bpp + Alpha 8 bpp;
- RGB 15 bpp + Extra 1 bpp;
- RGB 24 bpp + Extra 8 bpp;

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
AUTHOR_COMMENT	STRING	" "		Author's comments.
AUTHOR_NAME	STRING	" "		Name of author.
IMAGE_ID	STRING	" "		Image ID.
SOFTWARE_ID	STRING	" "		Software ID.
STAMP_DAY	WORD	0		Day stamp.
STAMP_HOUR	WORD	0		Hour stamp.
STAMP_MINUTE	WORD	0		Minute stamp.
STAMP_MONTH	WORD	0		Month stamp.
STAMP_SECOND	WORD	0		Second stamp.
STAMP_YEAR	WORD	0		Year stamp.
KEEP_ALPHA	BOOL	FALSE	TRUE, FALSE	FALSE means to ignore Alpha bits. Non-zero (TRUE) means to pass alpha bits back through callback.
PROMOTE16	BOOL	FALSE	TRUE, FALSE	FALSE means to create 16 bit DIB without promoting to 24 bits. Non-zero (TRUE) means promoting.
HEADER_TYPE	INT	0	1 or 2	1 or 2 for revision level of header.
STORES_TAMP	BOOL	FALSE	TRUE, FALSE	FALSE for no stamp, TRUE to save a stamp with image. HEADER_TYPE must be 2.
STAMP_WIDTH	UINT	0	0-64	Stamps width must not be larger than 64 pixels.
STAMP_HEIGHT	UINT	0	0-64	Stamps height must not be larger than 64 pixels.
PALETTE	BOOL	FALSE	TRUE, FALSE	TRUE indicates that color map should be created.
CONV_TO_16	BOOL	FALSE	TRUE, FALSE	If TRUE, convert to 16 bpp when saving.
THUMB_FLAG	UINT	0	TRUE, FALSE	TRUE if save should include a thumbnail.
THUMB_WIDTH	UINT	0		Width of thumbnail.
THUMB_HEIGHT	UINT	0		Height of thumbnail - Thumbnails are 24 bit.

Comments:

This file format was originally developed by AT&T for use with its image capture boards. The format was taken over by Truevision when it acquired the product line from AT&T. It is now commonly used for digitized images and also for high-quality images produced by ray tracers and other graphics applications.

It became a popular file format mainly because it was the first 24-bit truecolor format to come to the PC market. There are several varieties of Targa files; the most commonly used are the Targa 16, Targa 24, and Targa 32. The names are derived from the type of hardware used to create them.

The fixed-sized header information of the Targa format includes: the existence (or not) and colormap, location, size, pixel depth, image location, colormap (if it exists), and finally the image data itself.

Version 2.0 introduced a file footer that identifies it as the newest version and contains pointers to additional fields in two main structures: the "extension area" and the "developer directory". The extension area contains the addresses of many optional fields, one of the most popular being the "postage stamp image" (miniature of the main image).

The developer directory can be used to store proprietary information. Developers can register their own private fields with Truevision. A null-terminated ASCII string containing "TRUEVISION-XFILE.", and positioned at the end of the file, indicates that the footer is valid.

Targa defines 3 color methods: pseudo color, direct color, and truecolor. Pseudo color uses an index to a color palette. Direct color is like pseudo color except that the RGB components are looked up separately. In truecolor files, the color information is stored directly in the image data. The palettes used by Targa files are variable in size; they do not necessarily correlate to the bit depth of the image. The presence of a palette does not always mean that it is used to display the image.

Alpha channel handling

KEEP_ALPHA control parameter must be set to TRUE to enable loading and saving of Alpha/Extra channels from TGA images.

If bits 0..3 of Image Descriptor field are set to non-zero (additional channel is present), ImageGear treats additional bits as follows, depending on the Attributes Type field:

Attributes Type value	**Attributes type value meaning**	**Additional bits are loaded as:**
0	No Alpha Data included	Non-premultiplied Alpha (for incompliant images where Image Descriptor tells there is an additional channel but Attributes Type field is not set.)
1	Undefined data, can be ignored	Extra channel
2	Undefined data, should be retained	Extra channel
3	Non-premultiplied Alpha	Non-premultiplied Alpha
4	Premultiplied Alpha	Premultiplied Alpha
Other	Reserved or unassigned	Non-premultiplied Alpha

If bits 0..3 of Image Descriptor field are set to zero (no additional channel is present), ImageGear treats additional bits as Extra channel. This allows reading additional channel from images that have incorrect Image Descriptor field, but does not spoil image display if there is no meaningful data in the additional channel.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.51 TIFF

Full Name	TIFF (Tagged Image File Format)
Format ID	IG_FORMAT_TIF = 40
File Extension(s)	*.tif, *.tiff
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	Read, Write
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	Yes
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 6.0 - Added support for CMYK and YCbCr color images, and JPEG compression. Ability to store pixels in "tiles"
- Version 5.0 - Added ability to store palette color images and support for LZW compression. This version featured TIFF "classes."
- Version 4.0 - Added support for uncompressed RGB color images.
- Version 3.0 - First public release.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing
- IG_FLTR_MPAGEWRITEPSUPPORT - multi-page file writing
- IG_FLTR_PAGEDELETESUPPORT - page deleting from multi-page file
- IG_FLTR_PAGESWAPSUPPORT - page swapping in multi-page files
- IG_FLTR_MPDATASUPPORT - faster multi-page access by storing private format data (used only with IG_mpi_... and IG_mpf_... API)

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB - 1, 2, 4, 8 bpp;
  - Grayscale - 1, 2, 4, 8, 12, 16, 32 bpp;
  - RGB: 3, 6, 12, 24, 36, 48 bpp;
  - Lab: 3, 6, 12, 24, 36, 48 bpp;
  - CMYK: 4, 8, 16, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 2, 4, 8, 12, 16, 32 bpp;
  - RGB + Premultiplied Alpha: 4, 8, 16, 32, 48, 64 bpp;
  - Indexed RGB + Extra: 1, 2, 4, 8 bpc;
  - Lab + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - CMYK + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc.
- IG_COMPRESSION_PACKED_BITS:
  - Indexed RGB - 1, 2, 4, 8 bpp;
  - Grayscale - 1, 2, 4, 8, 12, 16, 32 bpp;
  - RGB: 3, 6, 12, 24, 36, 48 bpp;
  - Lab: 3, 6, 12, 24, 36, 48 bpp;
  - CMYK: 4, 8, 16, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 2, 4, 8, 12, 16, 32 bpp;
  - RGB + Premultiplied Alpha: 4, 8, 16, 32, 48, 64 bpp;
  - Indexed RGB + Extra: 1, 2, 4, 8 bpc;
  - Lab + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - CMYK + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc.
- IG_COMPRESSION_HUFFMAN:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_CCITT_G32D:
  - Indexed RGB: 1 bpp;
  - Grayscale: 1 bpp;
- IG_COMPRESSION_JPEG:
  - Grayscale: 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_DEFLATE:
  - Indexed RGB - 1, 2, 4, 8 bpp;
  - Grayscale - 1, 2, 4, 8, 12, 16, 32 bpp;
  - RGB: 3, 6, 12, 24, 36, 48 bpp;
  - Lab: 3, 6, 12, 24, 36, 48 bpp;
  - CMYK: 4, 8, 16, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 2, 4, 8, 12, 16, 32 bpp;
  - RGB + Premultiplied Alpha: 4, 8, 16, 32, 48, 64 bpp;
  - Indexed RGB + Extra: 1, 2, 4, 8 bpc;
  - Lab + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - CMYK + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc.
- IG_COMPRESSION_LZW:
  - Indexed RGB - 1, 2, 4, 8 bpp;
  - Grayscale - 1, 2, 4, 8, 12, 16, 32 bpp;
  - RGB: 3, 6, 12, 24, 36, 48 bpp;
  - Lab: 3, 6, 12, 24, 36, 48 bpp;
  - CMYK: 4, 8, 16, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 2, 4, 8, 12, 16, 32 bpp;

- RGB + Premultiplied Alpha: 4, 8, 16, 32, 48, 64 bpp;
- Indexed RGB + Extra: 1, 2, 4, 8 bpc;
- Lab + Extra: 1, 2, 4, 8, 12, 16 bpc;
- CMYK + Extra: 1, 2, 4, 8, 12, 16 bpc;
- Grayscale + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc;
- RGB + Premultiplied Alpha + Extra: 1, 2, 4, 8, 12, 16 bpc.

To use the LZW (Lempel-Ziv-Welch) compression scheme, attach the ImageGear LZW Component.

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 4, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 16, 24, 32, 64 bpp;
  - RGB + Premultiplied Alpha: 32, 48, 64 bpp;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
- IG_COMPRESSION_PACKED_BITS:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 4, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 16, 24, 32, 64 bpp;
  - RGB + Premultiplied Alpha: 32, 48, 64 bpp;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
- IG_COMPRESSION_HUFFMAN:
  - Indexed RGB: 1 bpp;
- IG_COMPRESSION_CCITT_G3:
  - Indexed RGB: 1 bpp;
- IG_COMPRESSION_CCITT_G4:
  - Indexed RGB: 1 bpp;
- IG_COMPRESSION_CCITT_G32D:
  - Indexed RGB: 1 bpp;
- IG_COMPRESSION_JPEG (Lossy, Progressive):
  - Grayscale: 8, 12 bpp;
  - RGB: 24, 36 bpp;
- IG_COMPRESSION_JPEG (Lossless):
  - Grayscale: 8, 16 bpp;
  - RGB: 24 bpp;
- IG_COMPRESSION_DEFLATE:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 4, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 16, 24, 32, 64 bpp;
  - RGB + Premultiplied Alpha: 32, 48, 64 bpp;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
- IG_COMPRESSION_LZW:
  - Indexed RGB: 1, 4, 8 bpp;
  - Grayscale: 8, 12, 16, 32 bpp;
  - RGB: 24, 36, 48 bpp;
  - Lab: 24, 36, 48 bpp;
  - CMYK: 4, 32, 48, 64 bpp;
  - Grayscale + Premultiplied Alpha: 16, 24, 32, 64 bpp;
  - RGB + Premultiplied Alpha: 32, 48, 64 bpp;
  - Lab + Extra: 8, 12, 16 bpc;
  - CMYK + Extra: 8, 12, 16 bpc;
  - Grayscale + Premultiplied Alpha + Extra: 8, 12, 16 bpc;
  - RGB + Premultiplied Alpha + Extra: 8, 12, 16 bpc;

To use the LZW (Lempel-Ziv-Welch) compression scheme, attach the ImageGear LZW Component.

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
BIG_ENDIAN	AT_BOOL	FALSE	FALSE, TRUE	If TRUE, big endian order is used for write operation ("MM"), in other case little endian is used("II").
BITONAL_PALETTE_MODE	enumTIFFBitonalPaletteMode	IG_TIF_BITONAL_PALETTE_MODE_LEGACY	An enumTIFFBitonalPaletteMode value	Specifies whether ImageGear shall fix strange looking palettes when reading bi-tonal TIFF images.
BUFFER_SIZE	DWORD	32768	Any DWORD>0	This parameter specifies the buffer size for each strip for write operation if WRITE_CONFIG= =IG_TIF_STRIP_FIXED_BUFFER
DATETIME	LPCHAR	""		Specifies value for DateTime (tag 306) to write into image.
DO_NOT_WRITE_PALETTE	AT_BOOL	FALSE	FALSE,TRUE	Set to TRUE to skip the palette when writing a TIFF.
DOCUMENT_NAME	LPCHAR	""	Any string	Specifies value for DocumentName (tag 269) to write into image.
FAST_PAGE_COUNT	AT_BOOL	FALSE	FALSE, TRUE	Affects page counting. When FALSE, ImageGear counts only those IFDs that contain images.

				When TRUE, ImageGear counts all IFDs, without checking for presence of images in them. The latter mode requires significantly less reading operations and thus works faster, especially if the image is accessed through a network.
FILL_ORDER	MODE	IG_FILL_MSB	IG_FILL_MSBIG_FILL_LSB	Specifies fill order (tag 266) for tiff file to be written.
IMAGE_BEFORE_IFD	AT_BOOL	FALSE	FALSE, TRUE	This flag specifies physical location of raster data inside TIFF file relatively to IFD record. If this value is TRUE then image data is to be written before IFD record.
IMAGE_HEIGHT	DWORD	0	Any positive value	Was used internally in previous versions of ImageGear.
IMAGE_WIDTH	DWORD	0	Any positive value	Was used internally in previous versions of ImageGear.
INCLUDE_PAGE_NUMBER	AT_BOOL	TRUE	FALSE, TRUE	If this parameter is TRUE then include tag 297 into TIFF image with real value of page number.
LOAD_FIRST_UNKNOWN_CHANNEL_AS_PALPHA	AT_BOOL	TRUE	TRUE, FALSE	This parameter specifies how to load first extra channel if ExtraSamples tag is missing. If TRUE, ImageGear loads first extra channel as Premultiplied Alpha channel. This mode provides support for RGBPA TIFF images written by earlier versions of ImageGear. If LoadFirstUnknownChannelAsPAlpha is FALSE, ImageGear loads all extra channels as extra channels.
MISSING_COMPRESSION	AT_MODE	0		Missing compression.
NEW_SUBFILE_TYPE	UINT	0xFF		This parameter specifies value for tag 254. If value of this control parameter 0xFF then default value is used according to TIFF 6.0 format specification.
NUMBER_OF_STRIPS	UINT	1	Any value >0	If WRITE_CONFIG= =IG_TIF_STRIP_FIXED_COUNT then this value is used as the number of strips to be written.
PHOTOMETRIC	UINT	IG_TIF_PHOTO_WHITEZERO	IG_TIF_PHOTO_WHITEZERO, IG_TIF_PHOTO_BLACKZERO, IG_TIF_PHOTO_RGB, IG_TIF_PHOTO_CMYK, IG_TIF_PHOTO_PALETTE, IG_TIF_PHOTO_TRANSPARENCY, IG_TIF_PHOTO_YCBCR, IG_TIF_PHOTO_CIELAB	Specifies photometric interpretation (tag 262) for write operation.
PLANAR	AT_BOOL	FALSE	FALSE, TRUE	Specifies tag value 284 for output image.
READ_JPEG_AS_YCBCR	AT_BOOL	FALSE	TRUE, FALSE	For internal use.
SAVE_DIFF_PREDICTOR	AT_BOOL	FALSE	FALSE, TRUE	If this parameter is TRUE, then output TIFF-LZW image will be produced using the horizontal differencing predictor.
SAVE_INDEXED_GRAY_AS_GRAY	AT_BOOL	TRUE	FALSE, TRUE	Affects saving of images that have grayscale or inverted grayscale palette. If TRUE then ImageGear saves the image with BlackZero or WhiteZero photometric interpretation. Otherwise, ImageGear saves the image as Paletted.
SAVE_IPTC_NAA	AT_BOOL	FALSE	TRUE, FALSE	Affects TIFF and EXIF-TIFF image saving. Set to TRUE to enable the saving of IPTC_NAA tag (33723) to the TIFF IFD, if IPTC metadata is provided by the metadata callbacks. If IPTC metadata is also provided as part of the PhotoshopResources metadata, then ImageGear overwrites it with a copy of metadata from IPTC_NAA tag. Set to TRUE to skip the writing of IPTC_NAA tag (for backward compatibility with ImageGear 17.1).
STITCH_TILES	AT_BOOL	FALSE	TRUE, FALSE	Set to TRUE to enable automatic tile stitching during image loading.
SUBIFD_PATH	LPCHAR	Empty string		Number of the SubIFD from which to load the image. If set to empty string (default), load image from root IFD. See "Camera Raw Image support" section for more detail.
TILE_H_COUNT	DWORD	10		If WRITE_CONFIG==IG_TIF_TILED_FIXED_COUNT then this value is used as number of tiles in horizontal dimension.
TILE_HEIGHT	DWORD			If WRITE_CONFIG==IG_TIF_TILED_FIXED_SIZE then this value is used to specify vertical dimension of each tile.
TILE_V_COUNT	DWORD	10		If WRITE_CONFIG==IG_TIF_TILED_FIXED_COUNT then this value is used as number of tiles in vertical dimension.
TILE_WIDTH	DWORD	64		If WRITE_CONFIG==IG_TIF_TILED_FIXED_SIZE then this value is used to specify horizontal dimension of each tile.
UPDATE_LUT16	AT_BOOL	TRUE	TRUE, FALSE	
UPDATE_PAGE_NUMBERS	AT_BOOL	TRUE	TRUE, FALSE	
WRITE_CLASS_F	AT_BOOL	FALSE	FALSE, TRUE	If this value is TRUE then image to be written in TIFF format compatible with class F requirements.
WRITE_CONFIG	MODE	IG_TIF_STRIP_FIXED_COUNT	IG_TIF_STRIP_FIXED_COUNT, IG_TIF_STRIP_FIXED_BUFFER, IG_TIF_TILED_FIXED_SIZE, IG_TIF_TILED_FIXED_COUNT	Specifies configuration of TIFF file to be written: • IG_TIF_STRIP_FIXED_COUNT - writes fixed number of strips. • IG_TIF_STRIP_FIXED_BUFFER - writes strips of size no more then given size. • IG_TIF_TILED_FIXED_SIZE - writes image in tiles of specified size. • IG_TIF_TILED_FIXED_COUNT - divides image into specified number of tiles vertically and horizontally.
WRITE70	AT_BOOL	TRUE	FALSE, TRUE	If this value is TRUE then output TIFF-JPEG

image will be produced in TIFF 7.0 compatible format but in other case it will be compatible with TIFF 6.0.

Comments:

TIFF was developed for use in storing black-and-white images from scanners and desktop publishing applications. Now, in its fourth release (version 6.0), it is one of the most detailed and versatile bitmap formats in use. It is supported by most art, imaging, and word-processing applications. It supports several compression schemes. Aside from saving image data in bitmap form, it can also contain vector or text-based images.

Containing just three fields, the TIFF header is simple and one of the shortest of all the graphics file format headers. But, the structure of a TIFF is complicated, with variable length fields, variable number of fields, and the ability to store information (other than the header) in any order desired.

The other two major components of the TIFF format are "Image File Directories" (IFDs) and the image or images themselves. There is one IFD per image stored. The combination of an IFD and an image is referred to as a "subfile". The header contains an offset pointer to the first IFD. If there are multiple IFDs, each contains an offset to the next. The last IFD contains a value that signifies the end of the file.

IFDs closely resemble a header structure, and the information stored in them is often referred to as "TIFF Header Information". Unlike a header, however, they contain a variable number of "tags" (pointers or fields). In addition, each tag can point to data with a variable length. TIFFs are notorious for the number of tags that they can contain, up to a maximum of 65,535 tags of nearly 100 different types (version 6.0). Tags are listed in order by code number so that a TIFF reader can easily determine what fields are present. While ImageGear reads and stores all TIFF tags, it utilizes a subset of all of the possible tags. See note on previous page.

In version 5.0, the presence of certain subgroups of tags determined the "class" to which the TIFF belonged. The classes are: TIFF-B-monochrome, TIFF-F-facsimile, TIFF-G-grayscale, TIFF-P-palette based, TIFF-R-RGB color, TIFF-X-any class, TIFF-Y-can use JPEG compression. Version 6.0 uses tags to divide the TIFF type into different file configurations, leaving behind the class concept. Version 6.0 configurations are: Bilevel, palette color, RGB, grayscale, YCbCr, and Class F (facsimile).

TIFF bitmap data can be stored in one of two configurations: strips or tiles. Strips are groups of adjoining rows of bitmap data, and can be found in version 5.0 and 6.0 files. Tiles were new to TIFF version 6.0. They are rectangular or square sections of bitmap data. The method of storage is determined in part by what kind of compression (if any) is used. JPEG compression can handle tiled images. Due to the need for padding with tile storage, tiling is not usually efficient for small images.

ImageGear supports the following compression schemes for TIFF:

- Uncompressed
- CCITT Group 3
- CCITT Group 3 2D
- CCITT Group 4
- Huffman
- JPEG
- Lossless JPEG
- LZW (Lempel-Ziv-Welch)
- Packbits
- Progressive JPEG
- Deflate

See the ImageGear Supported Compressions Reference for descriptions of these compression types. The compression tag of the IFD tells whether the image is compressed, and by what method. (Not all TIFF files can use JPEG compression. It is supported by version 6.0, but in version 5.0, only a "Y" class TIFF can use JPEG).

TIFF/EP

TIFF/EP format was designed to provide a means for storing "raw" (unprocessed) image from digital camera's sensor.

TIFF/EP allows you to store several versions of the same image in one file. Typically, TIFF/EP image includes a small preview and a raw image. It can also include a larger or full-size preview, or some other variations of the image.

TIFF/EP uses IFD trees for storing different versions of image. This is different from IFD chains that are used in baseline TIFF to store multiple pages.

ImageGear does not detect TIFF/EP as a separate file format. One of well known extensions to TIFF/EP is Adobe DNG format. ImageGear detects it as a separate file format.

Most of digital cameras store pixels in "mosaic" format. At a given pixel location either a Red, Green, Blue, Cyan, or some other color sample value is recorded. Such images are referred to as "Color Filter Array" type. TIFF/EP format uses a two-dimensional matrix called "Color Filter Array pattern" to describe positions of pixels of particular color in the mosaic image. In order to recreate the full color values in each pixel, it is necessary to interpolate intensities of neighboring pixels.

To enable loading images from TIFF SubIFDs, a new control parameter is added to TIFF filter: "SUBIFD_PATH". This parameter has the type of String, and its default value is "" (empty string). By default, ImageGear loads the image in the root IFD (thumbnail). If "SUBIFD_PATH" string begins with a number, then ImageGear loads the image from the corresponding SubIFD of the root IFD. For example, if SUBIFD_PATH is set to "3", ImageGear will load the image from the 3rd SubIFD.

SUBIFD_PATH parameter also affects metadata reading. Metadata is loaded starting from the IFD specified by SUBIFD_PATH.

ImageGear does not support color reconstruction of TIFF/EP images, unless they are detected to be a Adobe DNG image supported by ImageGear.

References Used:

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. "Graphic Image Format FAQ 3-4". James D. Murray, 1994-1996.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.52 TXT (ASCII Text)

Full Name	TXT (ASCII Text)
Format ID	IG_FORMAT_TXT = 41
File Extension(s)	*.txt
Data Type	Raster Image
Data Encoding	ASCII
Color Profile Support	No
ImageGear Multipage Support	Yes
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, MAC

ImageGear Supported Versions:

N/A

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_MPAGEREADPSUPPORT - multi-page file reading

ImageGear Read Support:

- IG_COMPRESSION_ASCII - Indexed RGB: 1 bpp

ImageGear Write Support:

No

ImageGear Filter Control Parameters:

Filter Control Parameter	Type	Default Value	Available Values	Description
XDPI	UINT	200		Horizontal resolution of image.
YDPI	UINT	200		Vertical resolution of image.
MARGINS_LEFT	LONG	1000		Left text margin on page, expressed as thousandths of an inch.
MARGINS_TOP	LONG	1000 (750 - for Unix)		Top text margin on page, expressed as thousandths of an inch.
MARGINS_RIGHT	LONG	1000		Right text margin on page, expressed as thousandths of an inch.
MARGINS_BOTTOM	LONG	1000 (750 - for Unix)		Bottom text margin on page, expressed as thousandths of an inch.
PAGE_WIDTH	DIMENSION	8500		Width of resulting page, expressed as thousandths of an inch.
PAGE_HEIGHT	DIMENSION	11000		Height of resulting page, expressed as thousandths of an inch.
POINT_SIZE	INT	10 (-1 - for Unix)		Font metric: If 0 then lines per page and character per line is used, else - specify font size.
WEIGHT	UINT	FALSE (0)	TRUE, FALSE	Font metric: if TRUE use bold font.
ITALIC	AT_BOOL	FALSE	TRUE, FALSE	Font metric: if TRUE use italic font.

TAB_STOP	UINT	3 (4 - for Unix)		The number of characters per tab.
TYPE_FACE	LP | CHAR	"\x00" ("courier" - for Unix)		Font metric:typeface name of the font, If empty string then default font used "Courier new".
LINES_PER_PAGE	UINT	0 (60 - for Unix)		Number of line per page.
CHAR_PER_LINE	UINT	0 (80 - for Unix)		Number of characters per line.
COMPATIBILITY_MODE	AT_BOOL	FALSE (TRUE - for Unix)	TRUE, FALSE	If TRUE use old algorithm, otherwise use ImageGear Algorithm.

Comments:

This is a widely used format for storing plain text files. ASCII data can also be used to give vector data instructions, but this is not supported by ImageGear.

The current, commonly used version of ASCII uses a 7-bit format and is known as "Full" or "Extended ASCII". The 128 (27) different data values include printable and non-printable values. The non-printable values are represented by the first 32 (0-31) values of ASCII, and are called "control values". They are used to communicate with screens or printers for placement of the characters. These control values represent tabs, line feeds, spaces, etc. Combinations of these values create "escape sequences" whose values are device-dependent upon implementation. To keep an ASCII file completely device-independent, a file usually does not contain any control values other than tab, line feed, and carriage return.

What makes a TXT file different from many bitmap formats is the byte order. A file is written in the natural order that it appears when output. There is no division into bit planes, or reverse order of bits and bytes. The eighth bit of each byte is normally set to zero. In older versions of TXT files, this was often used as a parity bit.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats, 2nd ed. Windcrest /McGraw-Hill, 1995.

1.2.6.7.53 WBMP

Full Name	WBMP (Wireless Bit-Map)
Format ID	IG_FORMAT_WBMP = 66
File Extension(s)	*.wbmp,
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 1.1

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE - Indexed RGB: 1 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

None

Comments:

The WBMP (Wireless Bit-Map) read/write format is optimized to support mobile computing devices that use the Wireless Application Protocol (WAP).

File contains small header with image parameters and array or pixels in uncompressed form.

References Used:

WAP WAE Specification Version, 24 May 1999.

1.2.6.7.54 WMF

Full Name	WMF (Windows Metafile Format)
Format ID	IG_FORMAT_WMF = 44
File Extension(s)	*.wmf,
Data Type	Metafile Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix (read raster portion only). MAC (read raster portion only)

ImageGear Supported Versions:

- Version 1 Metafiles prior to Windows 3.0
- Version 2 Metafiles for Windows 3.0 and later

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp
- IG_COMPRESSION_RLE:
  - Indexed RGB: 4, 8 bpp

ImageGear Filter Control Parameters:

Filter Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
UPSIDE_DOWN	BOOL		TRUE, FALSE	If TRUE then images will be saved upside-down.
TYPE	UINT		BMP_TYPE_BMC, BMP_TYPE_BMI, BMP_TYPE_BMI2	Type of the BMP file to be saved as part of WMF.
COMPRESSION	DWORD		BMP_COMP_RLE4, BMP_COMP_RLE8, BMP_COMP_RGB	Type of compression of BMP file.
TRUE_METAFILE	BOOL	TRUE (for Windows); FALSE (otherwise)	TRUE, FALSE	TRUE means executing of metafile commands (playing of metafile). Can be TRUE only for Windows. If the TRUE_METAFILE parameter is set to TRUE, the image will be opened as an RGB DIB for use with the GDI functions, which produce the image output. This

				causes the image to look like it is 24-bit per pixel, and 1024x1024 in dimensions. If the TRUE_METAFILE parameter is set to FALSE, then ImageGear will open the image according to its correct bit depth and dimensions.
RESOLUTION_X	DWORD	NULL		X resolution. 0 for actual resolution.
RESOLUTION_Y	DWORD	NULL		Y resolution. 0 for actual resolution.
DEPTH	DWORD	NULL		Bit Depth. 0 for actual depth.

Comments:

A Microsoft Windows Metafile holds vector and bitmap graphics data in memory or on disk. Although it was developed for use with Windows applications, it is now used by many non-Windows-based applications, allowing data to be transferred to and from Windows applications. Due to the great success of the Microsoft Windows interface, the Windows Metafile format is found in nearly all graphical applications. Metafiles use much less space and are more device-independent than bitmaps.

See also the section Support for Metafile Formats.

The Windows metafile begins with a short header and is followed by one or more records of data. The header describes the record data. A "placement" header can also be added before the file header; it contains information needed to move the metafile between applications. Each record corresponds to a binary-encoded Windows graphics device interface (GDI) call, and contains the size of the record, the unique function number for the GDI and an array of parameters. The GDI is used by Windows to perform all image output. When the metafile is "played", (this Microsoft term is a companion term to the Windows function named "PlayMetaFile"), each record makes a call to the appropriate function call for displaying each object in the image. The last record in the file contains a function number of zero to indicate that the end of the record data has been reached.

References Used:

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

Petzold, Charles. Programming Windows: The Microsoft Guide to writing applications for Windows 3. Redmond, WA: Microsoft Press, 1990.

1.2.6.7.55 WPG

Full Name	WPG (WordPerfect Graphics Metafile)
Format ID	IG_FORMAT_WPG = 42
File Extension(s)	*.wpg
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- WPG for WP 5.1 and up can store bitmap and vector graphics in the same file.
- WPG for WP 5.0 and prior can store only bitmap or vector graphic, but not both in same file.

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading

ImageGear Read Support:

- IG_COMPRESSION_RLE - Indexed RGB: 1, 4, 8 bpp

ImageGear Write Support:

None

ImageGear Filter Control Parameters:

None

Comments:

This format was created specifically for use with WordPerfect software products. WPG files for WordPerfect versions 5.1 and up can store both bitmap and vector image data in the same file.

The WordPerfect Graphics Metafile contains a short header or "prefix" (as WordPerfect Corporation referred to it). The header is followed by a record area, that is a sequence of objects and their attributes. The first record is called the "Start WPG Data" record and contains information on the size of the images and the version number of the .WPG file. The next record is usually a color map, unless the image is black and white. The next record is a bitmap record. If there are multiple images, there is a bitmap record for each image. The last record in a .WPG file contains a NULL body to signify the end of the file. These files may also contain Encapsulated PostScript (EPS) data.

References Used

Murray, James D. "Graphic Image Format FAQ 3-4". James D. Murray, 1994-1996.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.56 XBM

Full Name	XBM (X BitMap)
Format ID	IG_FORMAT_XBM = 43
File Extension(s)	*.xbm
Data Type	Raster Image
Data Encoding	ASCII
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 11, data stored as 1-byte character, 1986
- Version 10, data stored as 2-byte "short" integers

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_ASCII - Indexed RGB: 1 bpp

ImageGear Write Support:

- IG_COMPRESSION_ASCII - Indexed RGB: 1 bpp

ImageGear Filter Control Parameters:

None

Comments:

The XBM format is intended as a convenient storage method for small monochrome images, for example, cursor and icon bitmaps. It can, however, support images of any size, but since it supports no native compression scheme, an exterior compression program is used when compacting is desired. The bitmap data is stored as ASCII data with C language syntax, making the file easy to insert into C program code. XBM data can be stored as a standalone graphics file, or within a C program header file. See also XPM, XWD.

XBM files begin with two to four #define statements in substitution of a header. These identify the image width and height, and optionally, the coordinates of a Hotspot, if one exists.

The image data follows and is more free-form than the other bitmap data formats described in this chapter. It consists of one variable-length static array of pixel values. Each value (in version 11) consists of one byte of data, and therefore represents 8 1-bit pixels. The first pixel (0,0) is represented by the high bit of the first byte in the array. Due to this one-array format, there is nothing in the data that explicitly marks the rows of the bitmap.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.57 XPM

Full Name	XPM (X PixMap)
Format ID	IG_FORMAT_XPM = 45
File Extension(s)	*.xpm
Data Type	Raster Image
Data Encoding	ASCII (in C language syntax)
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 3.2g, April 1993

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp

ImageGear Filter Control Parameters:

None

Comments:

The XPM format was created as an extended version of the XBM file format. It is the informal standard for storing X Window pixmap data, including Hotspot information for cursor bitmaps. The image data is stored in ASCII text characters that are formatted as a standard C array of character strings. It is intended to be human-readable, is readily inserted into C/C++ program code, and can contain any number of comment lines. It therefore does not support a native compression scheme. If compacting is desired, an external compression program may be used. See also XBM, XWD.

All XPM files begin with a C language comment line containing "XPM". Following this are three sections of data: values, colors, and pixels, and an optional fourth section: extensions. The "values" section is the equivalent of the header structure typically found in a graphics file. It gives the size of the pixmap, as well as its number of colors, characters per pixel, the location of the Hotspot (if any), and an indicator of whether the file contains an extension section. Each section is set off with a comment-line title.

The colors section contains codes for the pixmap data characters. All pixels that make up the pixmap are assigned to one or more ASCII characters and one or more colors. (e.g. the character "X" may be assigned to the color red). There are several different conventions for identifying a color. If the string "None" appears as the color to be applied to a specific character, the character(s) symbolizes a transparent pixel.

The "pixels" section contains the bitmap data that appear as an array of character strings, where one row of bitmap data is represented by one array element. Each row is a group of characters set off by quotation marks. Each character is defined in the previous "colors" section.

If indicated by the values section, an extension section appears. It can contain one or more subsections that conform to one of two syntactical formats. An "XPMENDEXT" marker is always used to mark the end of the extension section.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.7.58 XWD

Full Name	XWD (X Window Dump)
Format ID	IG_FORMAT_XWD = 47
File Extension(s)	*.xwd, *.wd (for Unix)
Data Type	Raster Image
Data Encoding	Binary
Color Profile Support	No
ImageGear Multipage Support	No
ImageGear Alpha Channel Support	No
ImageGear Platforms Support	WIN32, WIN64, Unix, Unix64, .NET, .NET64, MAC

ImageGear Supported Versions:

- Version 7 for X11, June 1987 (X10 grayscale and palette only)

ImageGear Supported Features:

- IG_FLTR_DETECTSUPPORT - autodetection
- IG_FLTR_PAGEREADSUPPORT - single page file reading
- IG_FLTR_PAGEINSERTSUPPORT - single-page file writing

ImageGear Read Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Write Support:

- IG_COMPRESSION_NONE:
  - Indexed RGB: 1, 4, 8 bpp;
  - RGB: 24 bpp

ImageGear Filter Control Parameters:

None

Comments

An XWD file can contain a representation of the window, the background, or the entire screen display. It has been designed to be a very versatile, device-independent format. See also XBM, XPM.

The general structure of the XWD graphics format begins with a long header, that is sometimes followed by a palette and contains the bitmap data. The header contains integer data and stores the header size, the XWD version, the size and location of the bitmap, the window size, location, and border width. A ByteOrder field indicates whether the bytes are stored in big-endian or little-endian order.

In the interest of making this format device-independent, the XWD supports six "visual classes" and three image formats. The visual class code is stored in the visual_class field of the header, and represents the following categories:

- Static Gray, for most monochrome screens and using a fixed device-dependent color map;
- GrayScale, for monochrome screens and using a software-supplied palette;
- StaticColor, which uses a fixed device-dependent palette;
- Pseudocolor, which uses a software-supplied palette and is intended for VGA screens;
- TrueColor, with fixed device-dependent mapping of RGB values to screen colors;
- DirectColor, with software-supplied mapping of RGB values to screen values.

The image-format categories, whose codes are stored in the pixmap_format field of the header, are called XYBitmap (1-bit), XYPixmap (single plane), and ZPixmap (two or more planes).

Where the value of pixmap_format indicates GrayScale, PseudoColor or DirectColor, a palette follows the header.

The image data is the last structure in the file. The bytes are stored in rows with groupings called "units", whose lengths are determined by the bitmap_unit field of the header.

If the pixmap_format field is 1, indicating an XYPixmap, there are multiple representations of the bitmap data, one for each color plane, where the first bitmap represents the highest bit of the data; the second bitmap represents the second-highest bit, and so on. An image with a bit depth of 4 yields a file with four bitmaps.

References Used

Brown, C. Wayne, and Barry J. Shepherd. Graphics File Formats: Reference and Guide. Greenwich, CT.: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats. Sebastopol, CA: O'Reilly & Associates, Inc., 1994.

1.2.6.8 ImageGear Supported Non-Image Data Storage

This section provides detailed information about the following:

- Metadata Structure "ValueType" and "Value"
- Non-Image Data Structure

1.2.6.8.1 Metadata Structure "ValueType" and "Value"

Here are possible combinations of ValueType and Value elements of AT_DATALIST_ITEM ImageGear metadata structure:

ValueType	**Value**
AM_TID_META_INT8	● The "FLTR.METADATA_FORMAT" global parameter value is "text" : Value is pointer to NUL-terminated ASCII string - textual representation of "Length" 8-bit signed integers separated by comma. Example: "-128;0;+45;56" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" : Value is pointer to array of 8-bit signed integers of size "Length"
AM_TID_META_UINT8	● The "FLTR.METADATA_FORMAT" global parameter value is "text" : Value is pointer to NUL-terminated ASCII string - textual representation of "Length" 8-bit unsigned integers separated by comma. The hexadecimal values are allowed. Example: "0;+18;255;0xFA" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 8-bit signed integers of size "Length"
AM_TID_META_INT16	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NUL-terminated ASCII string - textual representation of "Length" 16-bit signed integers separated by comma. Example: "0;+1800;-255;32355" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 16-bit signed integers of size "Length"
AM_TID_META_UINT16	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NUL-terminated ASCII string - textual representation of "Length" 16-bit unsigned integers separated by comma. The hexadecimal values are allowed. Example: "0;+543;2550;0x12FF" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 16-bit signed integers of size "Length"
AM_TID_META_INT32	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NUL-terminated ASCII string - textual representation of "Length" 32-bit signed integers separated by comma. Example: "0;+67;-235987;32355" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 32-bit signed integers of size "Length"
AM_TID_META_UINT32	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NUL-terminated ASCII string - textual representation of "Length" 32-bit unsigned integers separated by comma. The hexadecimal values are allowed. Example: "0;+543;12362550;0x56FDE345" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 32-bit signed integers of size "Length"
AM_TID_META_INT64	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NUL-terminated string - textual representation of "Length" 64-bit signed integers separated by comma. ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 32-bit signed integers of size "Length"
AM_TID_META_UINT64	● The "FLTR.METADATA_FORMAT" global parameter value is "text":

	Value is pointer to NUL-terminated ASCII string - textual representation of "Length" 64-bit unsigned integers separated by comma. The hexadecimal values are allowed. ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 32-bit signed integers of size "Length"
AM_TID_META_BOOL	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NULL-terminated string - textual representation of "Length" Boolean values separated by comma. Example: "TRUE;FALSE;false;true" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 32-bit unsigned integers of size "Length" The value 1 represents boolean TRUE and the value 0 represents boolean FALSE.
AM_TID_META_RATIONAL_INT32	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NUL-terminated ASCII string - textual representation of "Length" rational values separated by comma. Each rational value is fraction where both numerator and denominator are 32-bit signed integers separated by slash. Zero is allowed. Example: "+324/-567;0/0;-68/45668" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 32-bit signed integers of size 2 * "Length" Each two elements represent pair (numerator, denominator) that represents rational number numerator/denominator.
AM_TID_META_RATIONAL_UNT32	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NUL-terminated ASCII string - textual representation of "Length" rational values separated by comma. Each rational value is fraction where both numerator and denominator are 32-bit unsigned integers separated by slash. Zero is allowed. Example: "+324/567;0/0" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 32-bit unsigned integers of size 2 * "Length" Each two elements represent pair (numerator, denominator) that represents rational number numerator/denominator.
AM_TID_META_FLOAT	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NUL-terminated ASCII string - textual representation of "Length" 4-byte float-point values separated by comma. Example: "-298.98676;3568732;6.9876E-10;0" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 4-byte float-point values of size "Length"
AM_TID_META_DOUBLE	● The "FLTR.METADATA_FORMAT" global parameter value is "text": Value is pointer to NUL-terminated ASCII string - textual representation of "Length" 8-byte (double precision) float-point values separated by comma. Example: "-29842.9867698098;356873245345;6.98766575489E+50;0" ● The "FLTR.METADATA_FORMAT" global parameter value is "binary" (DLL only): Value is pointer to array of 8-byte (double precision) float-point values of size "Length"
AM_TID_META_STRING	● It does not depend on "FLTR.METADATA_FORMAT" global parameter value Value is pointer to NUL-terminated ASCII string, "Length" - length of the string (NULL is not counted). Example: "Simple string"
AM_TID_RAW_DATA	● It does not depend on "FLTR.METADATA_FORMAT" global parameter value Value is pointer to byte array that represent binary data, "Length" - size of the array in bytes.

1.2.6.8.2 Non-Image Data Structure

In this section, the metadata items (and sequence of metadata items) are written in table form where metadata type constants are in the shorted form. For example, the LEVEL_START should be treated as IG_METAD_LEVEL_START, UINT32 - as AM_TID_META_UINT32, RAW_DAT - as AM_TID_RAW_DATA, etc.

- EXIF-JPEG Non-image Data Structure
- EXIF-TIFF Non-Image Data Structure
- GIF Non-image Data Structure
- IPTC Non-Image Data Structure
- JPEG Non-Image Data Structure
- PNG Non-Image Data Structure
- TIFF Non-Image Data Structure
- XMP Non-Image Data Structure

1.2.6.8.2.1 EXIF-JPEG Non-image Data Structure

The EXIF-JPEG metadata structure is similar to JPEG one. But EXIF-JPEG sends the Exif APP1 marker segment data in parsed form and Jfif APP0 marker segment data is not parsed and sent as raw data.

Brief information on EXIF-JPEG metadata levels is provided in the set of tables below:

- EXIF-JPEG Level
- EXIF Makernote
- Exif APP2 Marker Segment (Flashpix Extensions) Levels

EXIF-JPEG Level

All items between items with Name "EXIF" and Id IG_FORMAT_EXIF_JPEG (Type LEVEL_START and LEVEL_END) are interpreted as EXIF data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

For JPEG marker segment levels see JPEG Non-Image Data Structure.

Name	Id	Type	Value Type	Value	Value Length	Read Only
"EXIF-JPEG"	IG_FORMAT_EXIF_JPEG	LEVEL_START	0	NULL	0	TRUE
Exif APP1 marker segment level						
JPEG marker segment levels mixed in any way (if present)						
"EXIF-JPEG"	IG_FORMAT_EXIF_JPEG	LEVEL_END	0	NULL	0	TRUE

Exif APP1 Marker Segment Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"APP1"	65505	LEVEL_START	0	NULL	0	TRUE
"APP1_DATASIZE"	60225	VALUE_ITEM	UINT16	<Data>	1	TRUE
"EXIF_HEADER"	59935	VALUE_ITEM	STRING	"EXIF"	5	TRUE
"TIF_HEADER"	59936	VALUE_ITEM	UINT16	0x4949 or 0x4D4D	1	TRUE
IFD 0 level (if present)						
Thumbnail IFD level (if present)						
"APP1"	0xFFE1	LEVEL_END	0	NULL	0	TRUE

The "APP1_DATASIZE" item can be omitted during saving.

For more detailed Exif information see EXIF-TIFF Non-Image Data Structure.

EXIF Makernote

Makernote is a standard EXIF tag of UNDEFINED (byte) type. This tag usually is used as a "hidden" IFD. Makernote can not be read in and written out as a BLOB, because IFD offsets become invalid. Therefore, to preserve this tag while writing an image it's necessary to decode it during reading and re-encode it during writing, despite the fact that formally makernote is just a binary tag.

See enumIGEXIFMakerNoteType for descriptions of the various makernote types.

To present Makernote info in a convenient way ImageGear introduces a special structure as an addition to the standard EXIF Metadata: "Makernote*s* Wrapper IFD". Instead of a single binary Makernote*s* tag (37500), we are adding a virtual IFD, containing all the information about Makernote*s*.

'Makernote Wrapper IFD' has the following format depending on the 'type' tag:

1. Type IG_MAKERNOTE_TYPE_UNKNOWN:

```
'MakerNote wrapper IFD'
-------->'type' = IG_MAKERNOTE_TYPE_UNKNOWN
-------->'binary MakerNote'

Type IG_MAKERNOTE_TYPE_IFD:
'MakerNote wrapper IFD'
-------->'type' = IG_MAKERNOTE_TYPE_IFD
-------->'MakerNote IFD'
----------------> 'individual MakerNote tag 1'
----------------> 'individual MakerNote tag 2'
...
----------------> 'individual MakerNote tag N'
-------->'binary MakerNote'

Type IG_MAKERNOTE_TYPE_IFD_PREFIXED:
'MakerNote wrapper IFD'
```

```
-------->'type' = IG_MAKERNOTE_TYPE_IFD_PREFIXED
-------->'MakerNote IFD prefix'
-------->'MakerNote IFD'
----------------> 'individual MakerNote tag 1'
----------------> 'individual MakerNote tag 2'
...
----------------> 'individual MakerNote tag N'
-------->'binary MakerNote'

Type IG_MAKERNOTE_TYPE_TIF_HEADER_PREFIXED:
'MakerNote wrapper IFD'
-------->'type' = IG_MAKERNOTE_TYPE_TIF_HEADER_PREFIXED
<The rest of the structure is identical to IG_MAKERNOTE_TYPE_IFD_PREFIXED>

Type IG_MAKERNOTE_TYPE_IFD_PREFIXED_OFFSET_II:
'MakerNote wrapper IFD'
-------->'type' = IG_MAKERNOTE_TYPE_IFD_PREFIXED_OFFSET_II
<The rest of the structure is identical to IG_MAKERNOTE_TYPE_IFD_PREFIXED>
```

If 'Makernote' EXIF tag is present in the file then ImageGear will always provide it in its original binary form via meta-data callback ('binary Makernote' tag) on the read side, and will also optionally provide it as a decoded sub-IFD (w/ or w/o prefix depending on how it is stored in the original file) if it can be decoded.

ImageGear will ignore 'binary Makernote' tag on the write side for the 'Makernote wrapper IFDs' that have 'type' other than IG_MAKERNOTE_TYPE_UNKNOWN - and construct 'Makernote' EXIF tag based on the 'Makernote IFD'.

ImageGear will store 'binary Makernote' to file 'as is' for the 'Makernote wrapper IFDs' that have 'type' set to IG_MAKERNOTE_TYPE_UNKNOWN.

If 'Makernote wrapper IFD' is not provided to ImageGear during saving operation then 'Makernote' EXIF tag will not be saved into the output file.

Makernote IFD tags are listed below:

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"IFD"	IGMDTAG_ID_EXIF_MAKERNOTE	LEVEL_START	0	NULL	0	FALSE
"MakerNoteType"	IGMDTAG_ID_EXIF_MAKERNOTE_TYPE	VALUE_ITEM	UINT16	enumIGEXIFMakerNoteType	1	FALSE
"MakerNotePrefix"	IGMDTAG_ID_EXIF_MAKERNOTE_PREFIX	VALUE_ITEM	RAW_DATA			FALSE
"MakerNoteBinary"	IGMDTAG_ID_EXIF_MAKERNOTE_BINARY	VALUE_ITEM	RAW_DATA			FALSE
"IFD"	IGMDTAG_ID_EXIF_MAKERNOTE_DATA_IFD	LEVEL_START	0	NULL	0	FALSE
"UNDEFINED"		VALUE_ITEM	RAW_DATA			FALSE
...						
"IFD"	IGMDTAG_ID_EXIF_MAKERNOTE_DATA_IFD	LEVEL_END	0	NULL	0	FALSE
"IFD"	IGMDTAG_ID_EXIF_MAKERNOTE	LEVEL_END	0	NULL	0	FALSE

Custom Makernote Tags and IFDs

The following custom tags and IFDs are introduced for Makernote support:

Makernote Wrapper IFD: tag = 37500 (same as standard Makernote tag). This IFD is located in the "Exif IFD" (34665).

For information about new tags, which belong to the Makernote Wrapper IFD, see enumIGEXIFMakerNoteTagIDs.

Vendors and Models Currently Supported

Make	**Model**
Canon	Canon DIGITAL IXUS
Canon	Canon EOS D30
Canon	Canon PowerShot G2
Canon	Canon PowerShot S50
FUJIFILM	FinePix4900Z
LEICA	digilux 4.3
Minolta Co., Ltd.	DiMAGE 7i
Nikon	E5000 (TIF)
NIKON	E5700
NIKON	E950
OLYMPUS OPTICAL CO., LTD	C2040Z

OLYMPUS OPTICAL CO., LTD	C960Z,D460Z
Panasonic	DMC-LC5
RICOH	Caplio RR1
SANYO Electric Co., Ltd.	SR6
SANYO Electric Co., Ltd.	SX113
SANYO Electric Co., Ltd.	SX212
SEIKO EPSON CORP.	PhotoPC 850Z

Exif APP2 Marker Segment (Flashpix Extensions) Levels

EXIF file format allows you to store Flashpix extensions in APP2 marker segments.

FPXR Contents List APP2 Marker Segment

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"APP2"	0xFFE2	LEVEL_START	0	NULL	0	TRUE
"APP2_DATASIZE"	60226	VALUE_ITEM	UINT16		1	TRUE
"FPXR_HEADER"	61221	VALUE_ITEM	STRING	"FPXR"	4	FALSE
"FPXRVersion"	61222	VALUE_ITEM	UINT8		1	FALSE
"FPXRExtensionID"	61223	VALUE_ITEM	UINT8	"1"	1	FALSE
"FPXRInteroperabilityCount"	61224	VALUE_ITEM	UINT16	<n>	1	FALSE
Interoperability Entity level 0						
...						
Interoperability Entity level <n> - 1						
"APP2"	0xFFE2	LEVEL_END	0	NULL	0	TRUE

Interoperability Entity Level

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"InteroperabilityEntity"	<Ind.>	LEVEL_START	0	NULL	0	TRUE
"EntitySize"	<Ind.>	VALUE_ITEM	UINT32	0xFFFFFFFF for Storage or <Variable> for Stream	1	FALSE
"DefaultValue"	<Ind.>	VALUE_ITEM	UINT8	"FPXR"	4	FALSE
"Storage/StreamName"	<Ind.>	VALUE_ITEM	RAW_DATA	<Unicode name>	<Variable>	FALSE
The next item is present only if "EntitySize" item value is equal 0xFFFFFFFF (Storage)						
"EntityClassID"	<Ind.>	VALUE_ITEM	RAW_DATA		16	FALSE
"InteroperabilityEntity"	<Ind.>	LEVEL_END	0	NULL	0	TRUE

FPXR Data Stream APP2 Marker Segment

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"APP2"	0xFFE2	LEVEL_START	0	NULL	0	TRUE
"APP2_DATASIZE"	60226	VALUE_ITEM	UINT16		1	TRUE
"FPXR_HEADER"	61221	VALUE_ITEM	STRING	"FPXR"	4	FALSE
"FPXRVersion"	61222	VALUE_ITEM	UINT8		1	FALSE
"FPXRExtensionID"	61223	VALUE_ITEM	UINT8	"2"	1	FALSE
"FPXRIndexToContentsList"	61225	VALUE_ITEM	UINT16	<Ind.>*	1	FALSE
"FPXROffsetToStream"	61226	VALUE_ITEM	UINT32		1	FALSE
"FPXRStreamData"	61227	VALUE_ITEM	RAW_DATA		<Variable>	FALSE
"APP2"	0xFFE2	LEVEL_END	0	NULL	0	TRUE

*<Ind.> is pointer to appropriate "InteroperabilityEntity" of FPXR Contents List APP2 marker segment. <Ind.> value is between 0 and <n> - 1(<n> is the "FPXRInteroperabilityCount" item value of FPXR Contents List APP2 marker segment).

The Other FPXR APP2 Marker Segment

Name	Id	Type	Value Type	Value	Value Length	Read Only
"APP2"	0xFFE2	LEVEL_START	0	NULL	0	TRUE
"APP2_DATASIZE"	60226	VALUE_ITEM	UINT16		1	TRUE
"FPXR_HEADER"	61221	VALUE_ITEM	STRING	"FPXR"	4	FALSE
"FPXRVersion"	61222	VALUE_ITEM	UINT8		1	FALSE
"FPXRExtensionID"	61223	VALUE_ITEM	UINT8	More than "2"	1	FALSE
"FPXRData"	61228	VALUE_ITEM	RAW_DATA		<Variable>	FALSE
"APP2"	0xFFE2	LEVEL_END	0	NULL	0	TRUE

1.2.6.8.2.2 EXIF-TIFF Non-Image Data Structure

Brief information on EXIF-TIFF metadata levels is provided in the set of tables below:

- EXIF-TIFF Level
- IFD0 Level
- Tag Levels
- Description of Tags Used in EXIF
- Callback Required for Writing EXIF Metadata Items

EXIF-TIFF Level

The EXIF-TIFF metadata structure is similar to the TIFF metadata structure. However, EXIF-TIFF allows Exif subIFD tags, GPS subIFD tags and thumbnail IFD tags to be parsed and passed together with main IFD.

Name	Id	Type	Value Type	Value	Value Length	Read Only
"EXIF"	IG_FORMAT_EXIF_TIFF	LEVEL_START	0	NULL	0	TRUE
"TIF_HEADER"	59936	VALUE_ITEM	UINT16	0x4949 or 0x4D4D	1	TRUE
IFD 0 level (if present)						
Thumbnail IFD level (if present)						
"EXIF"	IG_FORMAT_EXIF_TIFF	LEVEL_END	0	NULL	0	TRUE

All items between items with Name "EXIF" and Id IG_FORMAT_EXIF_TIFF (Type LEVEL_START and LEVEL_END) are interpreted as EXIF data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

See Exif subIFD tags below for the list of Exif subIFD tag names and Ids.

IFD0 Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"IFD"	0	LEVEL_START	0	NULL	0	TRUE
Exif subIFD, GPS info subIFD and tags levels mixed in any way						
"IFD"	0	LEVEL_END	0	NULL	0	TRUE

Exif subIFD Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"IFD"	34665	LEVEL_START	0	NULL	0	TRUE
Interoperability info subIFD and tags levels mixed in any way						
"IFD"	34665	LEVEL_END	0	NULL	0	TRUE

The number 34665 is the Exif subIFD pointer tag (see below)

IFD1 (Thumbnail IFD), GPS Info subIFD and Interoperability Info subIFD Levels

Name	Id	Type	Value Type	Value	Value Length	Read Only
"IFD"	<IFDId>	LEVEL_START	0	NULL	0	TRUE
Tag 1						
...						
Tag n						
"IFD"	<IFDId>	LEVEL_END	0	NULL	0	TRUE

The value of <IFDId> is 1 for IFD1 or the subIFD pointer tag identifier for GPS (34853)(see GPS subIFD tags) or Interoperability (40965) subIFD (see Interoperability subIFD tags, below).

Tag Levels

For tag metadata structure see TIFF Non-Image Data Structure.

Description of Tags Used in EXIF

There are the following types of EXIF tags described in this section:

- Exif subIFD tags
- GPS subIFD tags
- Interoperability subIFD tags

For IFD0 and Thumbnail IFD tags see TIFF Non-Image Data Structure.

Exif subIFD Tags

The following table lists the most frequently used Exif tags. See enumIGEXIFTagIDs for a complete list of tags. For tags not listed in this table, to find out whether a tag is read only or not, see Non-Image Data Processing.

Item Name	**Item Id**	**Read Only**
"ExposureTime"	33434	FALSE
"Fnumber"	33437	FALSE
"ExposureProgram"	34850	FALSE
"SpectralSensitivity"	34852	FALSE
"ISOSpeedRatings"	34855	FALSE
"OECF"	34856	FALSE
"ExifVersion"	36864	FALSE
"DateTimeOriginal"	36867	FALSE
"DateTimeDigitized"	36868	FALSE
"ComponentsConfiguration"	37121	FALSE
"CompressedBitsPerPixel"	37122	FALSE
"ShutterSpeedValue"	37377	FALSE
"ApertureValue"	37378	FALSE
"BrightnessValue"	37379	FALSE
"ExposureBiasValue"	37380	FALSE
"MaxApertureValue"	37381	FALSE
"SubjectDistance"	37382	FALSE
"MeteringMode"	37383	FALSE
"LightSource"	37384	FALSE
"Flash"	37385	FALSE
"FocalLength"	37386	FALSE
"SubjectArea"	37396	FALSE
"MakerNote"	37500	FALSE
"UserComment"	37510	FALSE
"SubSecTime"	37520	FALSE
"SubSecTimeOriginal"	37521	FALSE
"SubSecTimeDigitized"	37522	FALSE
"FlashPixVersion"	40960	FALSE

"ColorSpace"	40961	FALSE
"PixelXDimension"	40962	TRUE
"PixelYDimension"	40963	TRUE
"RelatedSoundFile"	40964	FALSE
"InteroperabilityIFDPointer"	40965	TRUE
"FlashEnergy"	41483	FALSE
"SpatialFrequencyResponse"	41484	FALSE
"FocalPlaneXResolution"	41486	FALSE
"FocalPlaneYResolution"	41487	FALSE
"FocalPlaneResolutionUnit"	41488	FALSE
"SubjectLocation"	41492	FALSE
"ExposureIndex"	41493	FALSE
"SensingMethod"	41495	FALSE
"FileSource"	41728	FALSE
"SceneType"	41729	FALSE
"CFAPattern"	41730	FALSE
"CustomRendered"	41985	FALSE
"ExposureMode"	41986	FALSE
"WhiteBalance"	41987	FALSE
"DigitalZoomRatio"	41988	FALSE
"FocalLengthIn35mmFilm"	41989	FALSE
"SceneCaptureType"	41990	FALSE
"GainControl"	41991	FALSE
"Contrast"	41992	FALSE
"Saturation"	41993	FALSE
"Sharpness"	41994	FALSE
"DeviceSettingDescription"	41995	FALSE
"SubjectDistanceRange"	41996	FALSE
"ImageUniqueID"	42016	FALSE

GPS subIFD Tags

See enumIGEXIFGPSTagIDs for the complete list of EXIF GPS tags. All of EXIF GPS tags are writable.

Interoperability subIFD Tags

See enumIGEXIFInterOperTagIDs for the complete list of EXIF Interoperability tags. All of EXIF Interoperability tags are writable.

Callback Required for Writing EXIF Metadata Items

Value of these tags can be changed using LPAFT_IG_METAD_ITEM_SET_CB callback only.

Item Name	**Item Id**
"TIF_HEADER"	59936
IFD0 tags	
"PlanarConfiguration"	284
"YCbCrSubSampling"	530

"RowsPerStrip"	278
"YCbCrPositioning"	531
"XResolution"	282
"YResolution"	283
"ResolutionUnit"	296
Exif subIFD	
"ExifVersion"	36864
"FlashPixVersion"	40960
"ColorSpace"	40961
Thumbnail IFD	
"ImageWidth"	256
"ImageLength"	257
"PlanarConfiguration"	284
"YCbCrSubSampling"	530
"RowsPerStrip"	278
"YCbCrPositioning"	531
"XResolution"	282
"YResolution"	283
"ResolutionUnit"	296

The rest of metadata item values can be written using LPAFT_IG_METAD_ITEM_ADD_CB callback only.

1.2.6.8.2.3 GIF Non-image Data Structure

The GIF file format is complex and has different non-image data that can be stored before and after an image data. The GIF metadata design allows you to read/write any GIF non-image data and prevents misunderstanding with GIF extensions storing order.

The following metadata are always saved before an image data:

- GIF Logical Screen Descriptor Level
- GIF Image Descriptor Level
- GIF Global Color Table Level
- GIF Local Color Table Level

The metadata of GIF extensions that are inside GIF Extensions After Image Level (between items with Name "AfterImageExtensions" and Id 0xFFF (Type LEVEL_START and LEVEL_END)) are saved after an image data.

All other GIF Extensions Metadata are saved before an image data.

You can work with GIF metadata only when the ImageGear LZW Component is attached to the core ImageGear module.

Brief information on GIF metadata levels is provided in the set of tables below:

- GIF Metadata Level
- GIF Header Level
- GIF Logical Screen Descriptor Level
- GIF Global Color Table Level
- GIF Image Descriptor Level
- GIF Local Color Table Level
- GIF Extensions Metadata
- GIF Extensions After Image Level
- Callback Required for Writing GIF Metadata Items
- GIF Metadata Item ID Constants

GIF Metadata Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"GIF"	IG_FORMAT_GIF	LEVEL_START	0	NULL	0	TRUE
GIF header level						
GIF Logical Screen Descriptor level						
GIF Image Descriptor level						
optional Global and/or Local Color Tables and optional GIF Extensions levels (extensions before image)						
"GIF"	IG_FORMAT_GIF	LEVEL_END	0	NULL	0	TRUE

All items between items with Name "GIF" and Id IG_FORMAT_GIF (Type LEVEL_START and LEVEL_END) are interpreted as GIF data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

GIF Header Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"GIFHeader"	0x10	LEVEL_START	0	NULL	0	TRUE
"Signature"	0x101	VALUE_ITEM	STRING	"GIF"	3	TRUE
"Version"	0x102	VALUE_ITEM	STRING		3	TRUE
"GIFHeader"	0x10	LEVEL_END	0	NULL	0	TRUE

GIF Logical Screen Descriptor Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"LogicalScreenDescriptor"	0x20	LEVEL_START	0	NULL	0	TRUE
"LogicalScreenWidth"	0x201	VALUE_ITEM	UINT16		1	FALSE
"LogicalScreenHeight"	0x202	VALUE_ITEM	UINT16		1	FALSE
"Fields"	0x205	LEVEL_START	0	NULL	0	TRUE
"GlobalColorTableFlag"	0x206	VALUE_ITEM	BOOL		1	FALSE
"ColorResolution"	0x207	VALUE_ITEM	UINT8		1	FALSE
"SortFlag"	0x208	VALUE_ITEM	BOOL		1	FALSE
"GlobalColorTableSize"	0x209	VALUE_ITEM	UINT8		1	FALSE
"Fields"	0x205	LEVEL_END	0	NULL	0	TRUE
"BackgroundColorIndex"	0x203	VALUE_ITEM	UINT8			FALSE
"PixelAspectRatio"	0x204	VALUE_ITEM	UINT8			FALSE
"LogicalScreenDescriptor"	0x20	LEVEL_END	0	NULL	0	TRUE

GIF Global Color Table Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"GlobalColorTable"	0x30	VALUE_ITEM	RAW_DATA		Variable	TRUE

GIF Image Descriptor Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"ImageDescriptor"	0x2C	LEVEL_START	0	NULL	0	TRUE
"ImageLeftPosi tion"	0x2C1	VALUE_ITEM	UINT16		1	FALSE
"ImageTopPosition"	0x2C2	VALUE_ITEM	UINT16		1	FALSE
"ImageWidth"	0x2C3	VALUE_ITEM			1	FALSE
"ImageHeight"	0x2C4	VALUE_ITEM			1	FALSE
"Fields"	0x2C5	LEVEL_START	0	NULL	0	TRUE
"LocalColorTableFlag"	0x2C6	VALUE_ITEM	BOOL		1	FALSE
"InterlaceFlag"	0x2C7	VALUE_ITEM	BOOL		1	FALSE
"SortFlag"	0x2C8	VALUE_ITEM	BOOL		1	FALSE
"LocalColorTableSize"	0x2C9	VALUE_ITEM	UINT8		1	FALSE
"Fields"	0x2C5	LEVEL_END	0	NULL	0	TRUE
"ImageDescriptor"	0x2C	LEVEL_END	0	NULL	0	TRUE

GIF Local Color Table Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"LocalColorTable"	0x40	VALUE_ITEM	RAW_DATA		Variable	TRUE

GIF Extensions Metadata

GIF Graphic Control Extension Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"GraphicControlExtension"	0xF9	LEVEL_START	0	NULL	0	TRUE
"Fields"	0xF91	LEVEL_START	0	NULL	0	TRUE
"DisposalMethod"	0xF92	VALUE_ITEM	UINT8		1	FALSE
"UserInputFlag"	0xF93	VALUE_ITEM	BOOL		1	FALSE
"TransparentColorFlag"	0xF94	VALUE_ITEM	BOOL		1	FALSE
"Fields"	0xF91	LEVEL_END	0	NULL	0	TRUE
"DelayTime"	0xF95	VALUE_ITEM	UINT16		1	FALSE
"TransparentColorIndex"	0xF96	VALUE_ITEM	UINT8		1	FALSE
"GraphicControlExtension"	0xF9	LEVEL_END	0	NULL	0	TRUE

GIF Comment Extension Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"CommentExtension"	0xFE	VALUE_ITEM	STRING		Variable	TRUE

GIF Application Extension Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"ApplicationExtension"	0xFF	LEVEL_START	0	NULL	0	TRUE
"ApplicationIdentifier"	0xFF1	VALUE_ITEM	STRING		8	FALSE
"Appl.AuthenticationCode"	0xFF2	VALUE_ITEM	UINT8		4	FALSE
"ApplicationData"	0xFF3	VALUE_ITEM	RAW_DATA		Variable	FALSE
"ApplicationExtension"	0xFF	LEVEL_END	0	NULL	0	TRUE

GIF Plain Text Extension Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"PlainTextExtension"	0x01	LEVEL_START	0	NULL	0	TRUE
"TextGridLeftPosition"	0x11	VALUE_ITEM	UINT16		1	FALSE
"TextGridTopPosition"	0x12	VALUE_ITEM	UINT16		1	FALSE
"TextGridWidth"	0x13	VALUE_ITEM	UINT16		1	FALSE
"TextGridHeight"	0x14	VALUE_ITEM	UINT16		1	FALSE
"CharacterCellWidth"	0x15	VALUE_ITEM	UINT8		1	FALSE
"CharacterCellHeight"	0x16	VALUE_ITEM	UINT8		1	FALSE
"TextForegroundColorIndex"	0x17	VALUE_ITEM	UINT8		1	FALSE
"TextBackgroundColorIndex"	0x18	VALUE_ITEM	UINT8		1	FALSE
"PlainTextData"	0x19	VALUE_ITEM	STRING		Variable	FALSE

GIF Extensions After Image Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"AfterImageExtensions"	0xFFF	LEVEL_START	0	NULL	0	TRUE

One or more GIF extension levels						
"AfterImageExtensions"	0xFFF	LEVEL_END	0	NULL	0	TRUE

Callback Required for Writing GIF Metadata Items

The GIF Logical Screen Descriptor Level and GIF Image Descriptor Level metadata items can be written using LPAFT_IG_METAD_ITEM_SET_CB callback function.

Other GIF metadata can be written using LPAFT_IG_METAD_ITEM_ADD_CB callback function.

GIF Metadata Item ID Constants

Please see file enumIGGIFTagIDs for the complete list of GIF Metadata Item Id constants.

1.2.6.8.2.4 IPTC Non-Image Data Structure

IPTC, International Press and Telecommunications Council Standards, was created for exchanging different types of information associated with images.

IPTC is not a file format. In TIFF and EXIF_TIFF files, the IPTC data can be stored in a separate TIFF tag IPTC_NAA (id = 33723), or within Adobe Photoshop Resources (id = 34377). In JPEG and EXIF-JPEG files, the IPTC data is stored in Adobe Photoshop APP13 marker segment.

See TIFF Non-Image Data Structure, JPEG Non-Image Data Structure, and Photoshop Image Resource metadata structure.

Use TIFF control parameter SAVE_IPTC_NAA to control the saving of IPTC_NAA tag to TIFF and EXIF_TIFF formats.

Brief information on IPTC metadata levels is provided in the set of tables below:

- IPTC Level
- Record Level
- Dataset Value Item
- Dataset Value Items Description for IPTC Envelope Record (Record #1)
- Dataset Value Items Description for IPTC Application Record (Record #2)
- Dataset Value Items Description for IPTC Digital News Photo Parameter Record (Record #3)
- Dataset Value Items Description for IPTC Pre-Object Descriptor Record (Record #7)
- Dataset Value Items Description for IPTC Object Record (Record #8)
- Dataset Value Items Description for IPTC Post-Object Descriptor Record (Record #9)
- Callback Required for Writing IPTC Metadata Items

IPTC Level

Name	Id	Type	ValueType	Value	Value Length	Read Only
"IPTC"	0x1C00	LEVEL_START	UNDEFINED	NULL	0	TRUE
Record level 1						
...						
Record level n						
"IPTC"	0x1C00	LEVEL_END	UNDEFINED	NULL	0	TRUE

All items between items with Name "IPTC" and Id 0x1C00 (Type LEVEL_START and LEVEL_END) are interpreted as IPTC data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

Record Level

Name	Id	Type	ValueType	Value	Value Length	Read Only
"IPTC_RECORD"	<Record#>	LEVEL_START	UNDEFINED	NULL	0	TRUE
Dataset value item						
...						
Dataset value item						
"IPTC_RECORD"	<Record#>	LEVEL_END	UNDEFINED	NULL	0	TRUE

The<Record#> is identifier of IPTC record. Its value is the number of appropriate IPTC record described in IPTC - NAA IIM4 specification. Record levels must follow in numerical order within IPTC level.

Dataset Value Item

Name	Id	Type	ValueType	Value	Value Length	Read Only
<Dataset name>	<Dataset Id>	VALUE_ITEM	<Value Type>	<Data>	<Value Length>	FALSE

The available <Dataset name>, <Dataset Id>, <Value Length> and <Value Type> values are described below for each IPTC record according to IPTC-NAA IIM4 specification. Dataset value item can follow in any order within appropriate record level.

Dataset Value Items Description for IPTC Envelope Record (Record #1)

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
0	"ModelVersion"	AT_TID_WORD	1
5	"Destination"	AT_TID_STRING	1024
20	"FileFormat"	AT_TID_WORD	1
22	"FileFormatVersion"	AT_TID_WORD	1
30	"ServiceIdentifier"	AT_TID_STRING	10
40	"EnvelopeNumber"	AT_TID_STRING	8
50	"ProductID"	AT_TID_STRING	32
60	"EnvelopePriority"	AT_TID_STRING	1
70	"DateSent"	AT_TID_STRING	8
80	"TimeSent"	AT_TID_STRING	11
90	"CodedCharacterSet"	AT_TID_STRING	32
100	"UNO"	AT_TID_STRING	80
120	"ARMIdentifier"	AT_TID_WORD	1
122	"ARMVersion"	AT_TID_WORD	1

See also enumIGIPTCRecord1DatasetTags.

Dataset Value Items Description for IPTC Application Record (Record #2)

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
0	"RecordVersion"	AT_TID_WORD	1
3	"ObjectTypeReference"	AT_TID_STRING	67
4	"ObjectAttributeReference"	AT_TID_STRING	68
5	"ObjectName"	AT_TID_STRING	64
7	"EditStatus"	AT_TID_STRING	64
8	"EditorialUpdate"	AT_TID_STRING	2
10	"Urgency"	AT_TID_STRING	1
12	"SubjectReference"	AT_TID_STRING	236
15	"Category"	AT_TID_STRING	3
20	"SupplementalCategory"	AT_TID_STRING	32
22	"FixtureIdentifier"	AT_TID_STRING	32
25	"Keywords"	AT_TID_STRING	64
26	"ContentLocationCode"	AT_TID_STRING	3
27	"ContentLocationName"	AT_TID_STRING	64
30	"ReleaseDate"	AT_TID_STRING	8
35	"ReleaseTime"	AT_TID_STRING	11
37	"ExpirationDate"	AT_TID_STRING	8
38	"ExpirationTime"	AT_TID_STRING	11
40	"SpecialInstructions"	AT_TID_STRING	256

42	"ActionAdvised"	AT_TID_STRING	2
45	"ReferenceService"	AT_TID_STRING	10
47	"ReferenceDate"	AT_TID_STRING	8
50	"ReferenceNumber"	AT_TID_STRING	8
55	"DateCreated"	AT_TID_STRING	8
60	"TimeCreated"	AT_TID_STRING	11
62	"DigitalCreationDate"	AT_TID_STRING	8
63	"DigitalCreationTime"	AT_TID_STRING	11
65	"OriginatingProgram"	AT_TID_STRING	32
70	"ProgramVersion"	AT_TID_STRING	10
75	"ObjectCycle"	AT_TID_STRING	1
80	"By-line"	AT_TID_STRING	32
85	"By-lineTitle"	AT_TID_STRING	32
90	"City"	AT_TID_STRING	32
92	"Sublocation"	AT_TID_STRING	32
95	"Province/State"	AT_TID_STRING	32
100	"Country/ PrimaryLocationCode"	AT_TID_STRING	3
101	"Country/ PrimaryLocationName"	AT_TID_STRING	64
103	"OriginalTransmission Reference"	AT_TID_STRING	32
105	"Headline"	AT_TID_STRING	256
110	"Credit"	AT_TID_STRING	32
115	"Source"	AT_TID_STRING	32
116	"CopyrightNotice"	AT_TID_STRING	128
118	"Contact"	AT_TID_STRING	128
120	"Caption/Abstract"	AT_TID_STRING	2000
122	"Writer/Editor"	AT_TID_STRING	32
125	"RasterizedCaption"	AT_TID_BYTE	7360
130	"ImageType"	AT_TID_STRING	2
131	"ImageOrientation"	AT_TID_STRING	1
135	"LanguageIdentifier"	AT_TID_STRING	3
150	"AudioType"	AT_TID_STRING	2
151	"AudioSamplingRate"	AT_TID_STRING	6
152	"AudioSamplingResolution"	AT_TID_STRING	2
153	"AudioDuration"	AT_TID_STRING	6
154	"AudioOutcue"	AT_TID_STRING	64
200	"ObjectDataPreview FileFormat"	AT_TID_WORD	1
201	"ObjectDataPreview FileFormatVersion"	AT_TID_WORD	1
202	"ObjectDataPreviewData"	AT_TID_BYTE	Undefined

See also enumIGIPTCRecord2DatasetTags.

Dataset Value Items Description for IPTC Digital News Photo Parameter Record

(Record #3)

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
0	"RecordVersion"	AM_TID_TXT_UINT16	1
10	"PictureNumber"	AM_TID_RAW_DATA	16
20	"PixelsPerLine"	AM_TID_TXT_UINT16	1
30	"NumberOfLine"	AM_TID_TXT_UINT16	1
40	"PixelSizeInScanningDirection"	AM_TID_TXT_UINT16	1
50	"PixelSizePerpendicularTo ScanningDirection"	AM_TID_TXT_UINT16	1
55	"SupplementType"	AM_TID_TXT_UINT8	1
60	"ColourRepresentation"	AM_TID_TXT_UINT8	2
64	"InterchangeColourSpace"	AM_TID_TXT_UINT8	1
65	"ColourSequence"	AM_TID_TXT_UINT8	4
66	"ICCInputColourProfile"	AM_TID_RAW_DATA	Undefined
70	"ColourCalibrationMatrixTable"	AM_TID_RAW_DATA	Undefined
80	"LookupTable"	AM_TID_RAW_DATA	131072
84	"NumberOfIndexEntries"	AM_TID_TXT_UINT16	1
85	"ColourPalette"	AM_TID_RAW_DATA	Undefined
86	"NumberOfBitsPerSample"	AM_TID_TXT_UINT8	1
90	"SamplingStructure"	AM_TID_TXT_UINT8	1
100	"ScanningDirection"	AM_TID_TXT_UINT8	1
102	"ImageRotation"	AM_TID_TXT_UINT8	1
110	"DataCompressionMethod"	AM_TID_RAW_DATA	4
120	"QuantisationMethod"	AM_TID_TXT_UINT8	1
125	"EndPoints"	AM_TID_TXT_UINT8	Undefined
130	"ExcursionTolerance"	AM_TID_TXT_UINT8	1
135	"BitsPerComponent"	AM_TID_TXT_UINT8	Undefined
140	"MaximumDensityRange"	AM_TID_TXT_UINT16	1
145	"GammaCompensatedValue"	AM_TID_TXT_UINT16	1

See also enumIGIPTCRecord3DatasetTags.

Dataset Value Items Description for IPTC Pre-Object Descriptor Record (Record #7)

Item Identifier	Item Name	Value Type	Max. Value Length (for String last NULL is not counted)
10	"SizeMode"	AM_TID_TXT_UINT8	1
20	"MaxSubfileSize"	AM_TID_TXT_UINT8 or AM_TID_TXT_UINT16 or AM_TID_TXT_UINT32	1
90	"ObjectDataSizeAnnounced"	AM_TID_TXT_UINT8 or AM_TID_TXT_UINT16 or AM_TID_TXT_UINT32	1
95	"MaximumObjectDataSize"	AM_TID_TXT_UINT8 or AM_TID_TXT_UINT16 or AM_TID_TXT_UINT32	1

See also enumIGIPTCRecord7DatasetTags.

Dataset Value Items Description for IPTC Object Record (Record #8)

Item Identifier	**Item Name**	**Value Type**	**Max. Value Length (for String last NULL is not counted)**
10	"Subfile"	AM_TID_RAW_DATA	Undefined

See also enumIGIPTCRecord8DatasetTags.

Dataset Value Items Description for IPTC Post-Object Descriptor Record (Record #9)

Item Identifier	**Item Name**	**Value Type**	**Max. Value Length (for String last NULL is not counted)**
10	"ConfirmedObjectDataSize"	AM_TID_TXT_UINT8 or AM_TID_TXT_UINT16 or AM_TID_TXT_UINT32	1

See also enumIGIPTCRecord9DatasetTags.

Callback Required for Writing IPTC Metadata Items

All IPTC metadata items can be written using LPAFT_IG_METAD_ITEM_ADD_CB callback only.

1.2.6.8.2.5 JPEG Non-Image Data Structure

Brief information on JPEG metadata levels is provided in the set of tables below:

- JPEG Level
- JPEG Marker Segment Levels
- Frame Component Level
- Scan Component Level
- Define-Huffman-Tables Marker Segment Level
- JFIF APP0 Segment
- JFIF Extension (JFXX) APP0 Segment
- Photoshop Image Resource APP13 Marker Segment
- Callback Required for Writing JPEG Metadata Items
- JPEG Metadata Item Name and ID Constants

JPEG Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"JPEG-JFIF"	IG_FORMAT_JPG	LEVEL_START	0	NULL	0	TRUE
JPEG marker segment levels						
"JPEG-JFIF"	IG_FORMAT_JPG	LEVEL_END	0	NULL	0	TRUE

All items between items with Name "JPEG-JFIF" and Id IG_FORMAT_JPG (Type LEVEL_START and LEVEL_END) are interpreted as JPEG metadata. If during sending data from application level to filter level the first item is omitted the data will not be parsed and saved.

JPEG Marker Segment Levels

The following JPEG marker segments metadata are supported:

- Frame marker segment level - SOF0, SOF1, SOF2 and SOF3 (read only),
- Scan marker segment level - SOS (read only),
- Define-quantization-table marker segment level - DQT (read only),
- Define-Huffman-tables marker segment level - DHT (read only),
- Comment marker segment level - COM (read/write),
- Application marker segment level - APP0-APP15 (read/write).

Frame Marker Segment Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
<Frame marker name>	<Frame marker Id>	LEVEL_START	0	NULL	0	TRUE
<Frame marker size tag name>	<Frame marker size tag Id>	VALUE_ITEM	UINT16		1	TRUE
"Precision"	212	VALUE_ITEM	UINT8		1	TRUE
"Lines"	213	VALUE_ITEM	UINT16		1	TRUE
"SamplesPerLine"	214	VALUE_ITEM	UINT16		1	TRUE
"ComponentNumber"	215	VALUE_ITEM	UINT8	<n>	1	TRUE
Frame component level 1						
Frame component level <n>						
<Frame marker name>	<Frame marker Id>	LEVEL_END	0	NULL	0	TRUE

<Frame marker Id>, <Frame marker name>, <Frame marker size tag name> and <Frame marker size tag Id> are described in the table below:

Frame Marker	<Frame marker name>	<Frame marker Id>	<Frame marker size tag name>	<Frame marker size tag Id>
SOF0	"SOF0"	0xFFC0	"SOF0_DATASIZE"	60192
SOF1	"SOF1"	0xFFC1	"SOF1_DATASIZE"	60193
SOF2	"SOF2"	0xFFC2	"SOF2_DATASIZE"	60194
SOF3	"SOF3"	0xFFC3	"SOF3_DATASIZE"	60195

Frame Component Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"Component"	<Component No.>	LEVEL_START	0	NULL	0	TRUE
"Id"	<Component No.>	VALUE_ITEM	UINT		1	TRUE
"MCU_HV"	<Component No.>	VALUE_ITEM	UINT		1	TRUE
"QuantizationNumber"	<Component No.>	VALUE_ITEM	UINT		1	TRUE
"Component"	<Component No.>	LEVEL_END	0	NULL	0	TRUE

<Component No.> is frame component identifier of appropriate component.

Scan Marker Segment Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"SOS"	0xFFDA	LEVEL_START	0	NULL	0	TRUE
"SOS_DATASIZE"	60218	VALUE_ITEM	UINT16		1	TRUE
"ComponentNumber"	228	VALUE_ITEM	UINT8	<n>	1	TRUE
Scan component level 1						
Scan component level <n>						
"SpectralStart"	235	VALUE_ITEM	UINT8		1	TRUE
"SpectralEnd"	236	VALUE_ITEM	UINT8			TRUE
"AH_AL"	237	VALUE_ITEM	UINT8			TRUE
"SOS"	0xFFDA	LEVEL_START	0	NULL	0	TRUE

Scan Component Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"Component"	<Component No.>	LEVEL_START	0	NULL	0	TRUE
"Selector"	<Component No.>	VALUE_ITEM	UINT8		1	TRUE
"DC_AC"	<Component No.>	VALUE_ITEM	UINT8		1	TRUE
"Component"	<Component No.>	LEVEL_END	0	NULL	0	TRUE

<Component No.> is scan component selector of appropriate component.

Define-Quantization-Table Marker Segment Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"DQT"	0xFFDB	LEVEL_START	0	NULL	0	TRUE
"DQT_DATASIZE"	60219	VALUE_ITEM	UINT16		1	TRUE
"QuantizationTable"	<table 1 Id>	VALUE_ITEM	UINT8 or UINT16		64	TRUE

"QuantizationTable"	<table 1 Id>	VALUE_ITEM	UINT8 or UINT16		64	TRUE
"DQT"	0xFFDB	LEVEL_END	0	NULL	0	TRUE

<table Id> is the appropriate quantization table identifier.

Define-Huffman-Tables Marker Segment Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"DHT"	0xFFC4	LEVEL_START	0	NULL	0	TRUE
"DHT_DATASIZE"	60196	VALUE_ITEM	UINT16		1	TRUE
"DATA"	0xFFC4	VALUE_ITEM	RAW_DATA		<ValueSize>	TRUE
"DHT"	0xFFC4	LEVEL_END	0	NULL	0	TRUE

Comment Marker Segment Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"COM"	0xFFFE	LEVEL_START	0	NULL	0	TRUE
"COM_DATASIZE"	60254	VALUE_ITEM	UINT16		1	TRUE
"Comment"	0xFFFE	VALUE_ITEM	STRING		<ValueSize>	TRUE
"COM"	0xFFFE	LEVEL_END	0	NULL	0	TRUE

Application Marker Segment Level

There are several application marker segments, which data structure is well known. These segments are parsed and their data are passed in special format.

These marker segments are:

- JFIF APP0 segment
- JFIF extension (JFXX) APP0 segment (read only)
- EXIF APP1 segment
- Photoshop Image Resource APP13 marker segment (it includes some IPTC data and another Photoshop Image Resource metadata structure)
- Other application marker segment levels

JFIF APP0 Segment

Name	Id	Type	Value Type	Value	Value Length	Read Only
"APP0"	0xFFE0	LEVEL_START	0	NULL	0	TRUE
"APP0_DATASIZE"	60224	VALUE_ITEM	UINT16		1	FALSE
"JFIF_HEADER"	200	VALUE_ITEM	STRING	JFIF	5	TRUE
"Version"	201	VALUE_ITEM	UINT16		1	FALSE
"ResolutionUnits"	202	VALUE_ITEM	UINT8		1	FALSE
"ResolutionX"	203	VALUE_ITEM	UINT16		1	FALSE
"ResolutionY"	204	VALUE_ITEM	UINT16		1	FALSE
"ThumbnailWidth"[1]	205	VALUE_ITEM	UINT8		1	TRUE
"ThumbnailHeight"[2]	206	VALUE_ITEM	UINT8		1	TRUE
"APP0"	0xFFE0	LEVEL_END	0	NULL	0	TRUE

[1]The thumbnail width tags are present only if the thumbnail is stored to JFIF segment.

[2]The thumbnail height tags are present only if the thumbnail is stored to JFIF segment.

JFIF Extension (JFXX) APP0 Segment

Name	Id	Type	Value Type	Value	Value Length	Read Only
"APP0"	0xFFE0	LEVEL_START	0	NULL	0	TRUE
"APP0_DATASIZE"	60224	VALUE_ITEM	UINT16		1	TRUE
"JFIF_EX_HEADER"	209	VALUE_ITEM	STRING	"JFXX"	5	TRUE
"ExtensionCode"	210	VALUE_ITEM	UINT8		1	TRUE
"APP0"	0xFFE0	LEVEL_END	0	NULL	0	TRUE

Photoshop Image Resource APP13 Marker Segment

Name	Id	Type	Value Type	Value	Value Length	Read Only
"APP13"	0xFFED	LEVEL_START	0	NULL	0	TRUE
"APP13_DATASIZE"	60237	VALUE_ITEM	UINT16		1	FALSE
"PHOTOSHOP_HEADER"	59915	VALUE_ITEM	STRING	"Photoshop 3.0"	14	FALSE
Photoshop Image Resource level 1						
...						
Photoshop Image Resource level n						
"APP13"	0xFFED	LEVEL_END	0	NULL	0	TRUE

For "Photoshop Image Resource" metadata structure see in TIFF non-image data the Photoshop Image Resource metadata structure table.

Other Application Marker Segment Levels

Name	Id	Type	Value Type	Value	Value Length	Read Only
<Marker name>	<Marker Id>	LEVEL_START	0	NULL	0	TRUE
<Marker size tag name>	<Marker size tag Id>	VALUE_ITEM	UINT16		1	FALSE
"DATA"	<Marker Id>	VALUE_ITEM	RAW_DATA		<ValueSize>	FALSE
<Marker name>	<Marker Id>	LEVEL_END	0	NULL	0	TRUE

<Marker Id>, <Marker name>, <Marker size tag name> and <Marker size tag Id> are in table below:

Marker	<Marker name>	<Marker Id>	<Marker size tag name>	<Marker size tag Id>
APP0	"APP0"	0xFFE0	"APP0_DATASIZE"	60224
APP1	"APP1"	0xFFE1	"APP1_DATASIZE"	60225
APP2	"APP2"	0xFFE2	"APP2_DATASIZE"	60226
APP3	"APP3"	0xFFE3	"APP3_DATASIZE"	60227
APP4	"APP4"	0xFFE4	"APP4_DATASIZE"	60228
APP5	"APP5"	0xFFE5	"APP5_DATASIZE"	60229
APP6	"APP6"	0xFFE6	"APP6_DATASIZE"	60230
APP7	"APP7"	0xFFE7	"APP7_DATASIZE"	60231
APP8	"APP8"	0xFFE8	"APP8_DATASIZE"	60232
APP9	"APP9"	0xFFE9	"APP9_DATASIZE"	60233
APP10	"APP10"	0xFFEA	"APP10_DATASIZE"	60234
APP11	"APP11"	0xFFEB	"APP11_DATASIZE"	60235
APP12	"APP12"	0xFFEC	"APP12_DATASIZE"	60236

APP13	"APP13"	0xFFED	"APP13_DATASIZE"	60237
APP14	"APP14"	0xFFEE	"APP14_DATASIZE"	60238
APP15	"APP15"	0xFFEF	"APP15_DATASIZE"	60239

Callback Required for Writing JPEG Metadata Items

All JPEG metadata items can be written using LPAFT_IG_METAD_ITEM_ADD_CB callback only.

JPEG Metadata Item Name and ID Constants

Please see enumIGJPGTagIDs for a complete list of JPEG Metadata Id constants.

1.2.6.8.2.6 PNG Non-Image Data Structure

Brief information on PNG metadata levels is provided in the set of tables below:

- PNG Metadata Level
- PNG Chunks Levels
- Callback Required for Writing PNG Metadata Items
- PNG Metadata ID Constants

PNG Metadata Level

All items between items with Name "PNG" and Id IG_FORMAT_PNG (Type LEVEL_START and LEVEL_END) are interpreted as PNG data. If during sending data from application level to filter level the first item is omitted the data will not be interpreted and saved.

Name	Id	Type	Value Type	Value	Value Length	Read Only
"PNG"	IG_FORMAT_PNG	LEVEL_START	0	NULL	0	TRUE
PNG chunks levels						
"PNG"	IG_FORMAT_PNG	LEVEL_END	0	NULL	0	TRUE

PNG Chunks Levels

The following PNG non-image chunk levels are supported by ImageGear:

- PNG Header Chunk Metadata
- Physical Dimension Chunk Metadata
- Transparency Chunk Metadata
- Gamma Chunk Metadata
- Primary Chromaticities Chunk Metadata
- sRGB Chunk Metadata
- ICC profile Chunk Metadata
- Background Chunk Metadata
- Significant Bits Chunk Metadata
- Suggested Palette Chunk Metadata
- Palette Histogram Chunk Metadata
- Modified Time Chunk Metadata
- Text Chunk Metadata
- Compressed Textual Data Chunk Metadata
- International Textual Data Chunk Metadata
- Calibration of Pixel Values Chunk Metadata
- Physical Scale Chunk Metadata
- GIF Application Extension Chunk Metadata
- GIF Graphic Control Extension Chunk Metadata
- Image Offset Chunk Metadata
- The Rest Chunk Metadata

PNG Header Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"IHDR"	0x49484452	LEVEL_START	0	NULL	0	TRUE
"Width"	0x49484452	VALUE_ITEM	UINT32		1	TRUE
"Height"	0x49484452	VALUE_ITEM	UINT32		1	TRUE
"BitDepth"	0x49484452	VALUE_ITEM	UINT8		1	TRUE
"ColorType"	0x49484452	VALUE_ITEM	UINT8		1	TRUE
"CompressionType"	0x49484452	VALUE_ITEM	UINT8		1	TRUE

"FilterType"	0x49484452	VALUE_ITEM	UINT8		1	TRUE
"InterlaceType"	0x49484452	VALUE_ITEM	UINT8		1	TRUE
"IHDR"	0x49484452	LEVEL_END	0	NULL	0	TRUE

Physical Dimension Chunk Metadata

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"pHYs"	0x70485973	LEVEL_START	0	NULL	0	TRUE
"XAxis"	0x70485973	VALUE_ITEM	UINT32		1	FALSE
"YAxis"	0x70485973	VALUE_ITEM	UINT32		1	FALSE
"UnitSpecifier"	0x70485973	VALUE_ITEM	UINT8		1	FALSE
"pHYs"	0x70485973	LEVEL_END	0	NULL	0	TRUE

Transparency Chunk Metadata

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"tRNS"	0x74524e53	VALUE_ITEM	UINT8 or UINT16		Variable	FALSE

Gamma Chunk Metadata

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"gAMA"	0x67414d41	VALUE_ITEM	UINT32		1	FALSE

Primary Chromaticities Chunk Metadata

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"cHRM"	0x6348524d	LEVEL_START	0	NULL	0	TRUE
"WhitePoint"	0x6348524d	LEVEL_START	0	NULL	0	TRUE
"x"	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
"y"	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
"WhitePoint"	0x6348524d	LEVEL_END	0	NULL	0	TRUE
"Red"	0x6348524d	LEVEL_START	0	NULL	0	TRUE
"x"	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
"y"	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
"Red"	0x6348524d	LEVEL_END	0	NULL	0	TRUE
"Green"	0x6348524d	LEVEL_START	0	NULL	0	TRUE
"x"	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
"y"	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
"Green"	0x6348524d	LEVEL_END	0	NULL	0	TRUE
"Blue"	0x6348524d	LEVEL_START	0	NULL	0	TRUE
"x"	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
"y"	0x6348524d	VALUE_ITEM	UINT32		1	FALSE
"Blue"	0x6348524d	LEVEL_END	0	NULL	0	TRUE
"cHRM"	0x6348524d	LEVEL_END	0	NULL	0	TRUE

sRGB Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"sRGB"	0x73524742	VALUE_ITEM	UINT32		1	FALSE

ICC profile Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"iCCP"	0x70485973	LEVEL_START	0	NULL	0	TRUE
"ProfileName"	0x70485973	VALUE_ITEM	STRING		Variable	FALSE
"ProfileData"	0x70485973	VALUE_ITEM	RAW_DATA		Variable	FALSE
"iCCP"	0x70485973	LEVEL_END	0	NULL	0	TRUE

Background Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"bKGD"	0x624b4744	VALUE_ITEM	UINT8 or UINT16		Variable	FALSE

Significant Bits Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"sBIT"	0x73424954	VALUE_ITEM	UINT8		Variable	FALSE

Suggested Palette Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"sPLT"	0x73504c54	LEVEL_START	0	NULL	0	TRUE
"PaletteName"	0x73504c54	VALUE_ITEM	STRING		Variable	TRUE
"SampleDepth"	0x73504c54	VALUE_ITEM	UINT8		1	FALSE
"PaletteEntryCount"	0x73504c54	VALUE_ITEM	UINT16	<N>	1	FALSE
Palette entry level 0						
...						
Palette entry level n						
"sPLT"	0x70485973	LEVEL_END	0	NULL	1	FALSE

Palette Entry Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"PaletteEntry"	<Entry No.>	LEVEL_START	0	NULL	0	TRUE
"Red"	<Entry No.>	VALUE_ITEM	UINT8 or UINT16		1	FALSE
"Green"	<Entry No.>	VALUE_ITEM	UINT8 or UINT16		1	FALSE
"Blue"	<Entry No.>	VALUE_ITEM	UINT8 or UINT16		1	FALSE
"Alpha"	<Entry No.>	VALUE_ITEM	UINT8 or UINT16		1	FALSE
"Frequency"	<Entry No.>	VALUE_ITEM	UINT16		1	FALSE
"PaletteEntry"	<Entry No.>	LEVEL_END	0	NULL	0	TRUE

Palette Histogram Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"hIST"	0x68495354	VALUE_ITEM	UINT16		Variable	FALSE

Modified Time Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"tIME"	0x74494d45	LEVEL_START	0	NULL	0	TRUE
"Year"	0x74494d45	VALUE_ITEM	UINT16		1	FALSE
"Month"	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
"Day"	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
"Hour"	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
"Minute"	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
"Second"	0x74494d45	VALUE_ITEM	UINT8		1	FALSE
"tIME"	0x74494d45	LEVEL_END	0	NULL	0	TRUE

Text Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"tEXt"	0x74455874	LEVEL_START	0	NULL	0	TRUE
"Keyword"	0x74455874	VALUE_ITEM	STRING		Variable	FALSE
"Text"	0x74455874	VALUE_ITEM	STRING		Variable	FALSE
"tEXt"	0x74455874	LEVEL_END	0	NULL	0	TRUE

Compressed Textual Data Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"zTXt"	0x7a545874	LEVEL_START	0	NULL	0	TRUE
"Keyword"	0x7a545874	VALUE_ITEM	STRING		Variable	FALSE
"CompressionMethod"	0x7a545874	VALUE_ITEM	UINT8		1	FALSE
"Text"	0x7a545874	VALUE_ITEM	STRING		Variable	FALSE
"zTXt"	0x7a545874	LEVEL_END	0	NULL	0	TRUE

International Textual Data Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"zTXt"	0x69545874	LEVEL_START	0	NULL	0	TRUE
"Keyword"	0x69545874	VALUE_ITEM	STRING		Variable	FALSE
"CompressionFlag"	0x69545874	VALUE_ITEM	BOOL			
"CompressionMethod"	0x69545874	VALUE_ITEM	UINT8		1	FALSE
"LanguageTag"	0x69545874	VALUE_ITEM	STRING			
"TranslatedKeyword"	0x69545874	VALUE_ITEM	RAW_DATA			
"Text"	0x69545874	VALUE_ITEM	RAW_DATA		Variable	FALSE
"zTXt"	0x69545874	LEVEL_END	0	NULL	0	TRUE

Calibration of Pixel Values Chunk Metadata

Name	Id	Type	Value Type	Value	Value Length	Read Only
"pCAL"	0x7043414c	LEVEL_START	0	NULL	0	TRUE
"CalibrationName"	0x7043414c	VALUE_ITEM	STRING		Variable	FALSE
"OriginalZero(x0)"	0x7043414c	VALUE_ITEM	INT32		1	FALSE

"OriginalMax(x1)"	0x7043414c	VALUE_ITEM	INT32		1	FALSE
"EquationType"	0x7043414c	VALUE_ITEM	UINT8		1	FALSE
"NumberOfParameters"	0x7043414c	VALUE_ITEM	UINT8	<N>	Variable	FALSE
"UnitName"	0x7043414c	VALUE_ITEM	STRING		Variable	FALSE
"CalibrationParameter"	0	VALUE_ITEM	STRING		Variable	FALSE
...						
"CalibrationParameter"	<N> - 1	VALUE_ITEM	STRING		Variable	FALSE
"pCAL"	0x7043414c	LEVEL_END	0	NULL	0	TRUE

Physical Scale Chunk Metadata

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"sCAL"	0x7343414c	LEVEL_START	0	NULL	0	TRUE
"UnitSpecifier"	0x7343414c	VALUE_ITEM	UINT8		1	FALSE
"PixelWidth"	0x7343414c	VALUE_ITEM	STRING		Variable	FALSE
"PixelHeigt"	0x7343414c	VALUE_ITEM	STRING		Variable	FALSE
"sCAL"	0x7343414c	LEVEL_END	0	NULL	0	TRUE

GIF Application Extension Chunk Metadata

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"gIFx"	0x67494678	LEVEL_START	0	NULL	0	TRUE
"ApplicationIdentifier"	0x67494678	VALUE_ITEM	STRING		Variable	FALSE
"AuthenticationCode"	0x67494678	VALUE_ITEM	UINT8		1	FALSE
"ApplicationData"	0x67494678	VALUE_ITEM	RAW_DATA		Variable	FALSE
"gIFx"	0x67494678	LEVEL_END	0	NULL	0	TRUE

GIF Graphic Control Extension Chunk Metadata

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"gIFg"	0x67494674	LEVEL_START	0	NULL	0	TRUE
"DisposalMethod"	0x67494674	VALUE_ITEM	UINT8		1	FALSE
"UserInputFlag"	0x67494674	VALUE_ITEM	UINT8		1	FALSE
"DelayTime"	0x67494674	VALUE_ITEM	UINT16		1	FALSE
"gIFg"	0x67494674	LEVEL_END	0	NULL	0	TRUE

Image Offset Chunk Metadata

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"oFFs"	0x6f464673	LEVEL_START	0	NULL	0	TRUE
"XPosition"	0x6f464673	VALUE_ITEM	UINT32		1	FALSE
"YPosition"	0x6f464673	VALUE_ITEM	UINT32		1	FALSE
"UnitSpecifier"	0x6f464673	VALUE_ITEM	UINT8		1	FALSE
"oFFs"	0x6f464673	LEVEL_END	0	NULL	0	TRUE

The Rest Chunk Metadata

The rest chunk data are passed as raw data

Name	Id	Type	Value Type	Value	Value Length	Read Only
<Chunk Name>	<Chunk ID>	VALUE_ITEM	RAW_DATA		Variable	FALSE

<Chunk Name> is a string representation of the chunk type value.

<Chunk ID> is a binary representation of the chunk type value.

Callback Required for Writing PNG Metadata Items

Value of these metadata items can be changed using LPAFT_IG_METAD_ITEM_SET_CB callback only:

Item Name	Item Id
Physical dimension chunk metadata	
"XAxis"	0x70485973
"YAxis"	0x70485973
"UnitSpecifier"	0x70485973

The rest of the PNG metadata items can be written using LPAFT_IG_METAD_ITEM_ADD_CB callback only.

PNG Metadata ID Constants

Please see enumIGPNGTagIDs for the complete list of PNG Metadata Item Id constants.

1.2.6.8.2.7 TIFF Non-Image Data Structure

Brief information on TIFF metadata levels is provided in the set of tables below:

- TIFF Level
- IFD Level
- Tags
- Callback Required for Writing TIFF Metadata Items

TIFF Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"TIFF"	IG_FORMAT_TIF	LEVEL_START	0	NULL	0	TRUE
"TIF_ HEADER"	59936	VALUE_ITEM	UINT16	0x4949 or 0x4D4D	1	TRUE
Main IFD level						
"TIFF"	IG_FORMAT_TIF	LEVEL_END	0	NULL	0	TRUE

All items between items with Name "TIFF" and IdIG_FORMAT_TIF (Type LEVEL_START and LEVEL_END) are interpreted as TIFF data. If during sending data from application level to filter level the first item is omitted the data will not be parsed and saved.

The subIFDs is not parsed and passed.

IFD Level

Name	Id	Type	Value Type	Value	Value Length	Read Only
"IFD"	0	LEVEL_START	0	NULL	0	TRUE
One or more tag items						
"IFD"	0	LEVEL_END	0	NULL	0	TRUE

Tags

Most of TIFF tags are passed through metadata callback as

Name	Id	Type	Value Type	Value	Value Length	Read Only
<TagName>	<TagId>	VALUE_ITEM	<ValueType>	<TagValue>	<TagCount>	<R.o.>

The valid values of <TagName>, <TagId> and "Read Only" attributes are available in Description of TIFF tags Table.

See Also:TIFF Complex Data Tags

It is possible to use tag identifier and value of nonstandard tag (user defined tag).

The <TagCount> is real tag count value that is read/written in file.

The < ValueType > value is the metadata type constant that matches TIFF tag type as described in the table below:

TIFF Tag Type	Metadata Type
BYTE	AM_TID_META_UINT8
SBYTE	AM_TID_META_INT8
SHORT	AM_TID_META_UINT16
SSHORT	AM_TID_META_INT16
LONG	AM_TID_META_UINT32
SLONG	AM_TID_META_INT32
RATIONAL	AM_TID_META_RATIONAL_UINT32
SRATIONAL	AM_TID_META_RATIONAL_INT32

FLOAT	AM_TID_META_FLOAT
DOUBLE	AM_TID_META_DOUBLE
ASCII	AM_TID_META_STRING
UNDEFINED	AM_TID_RAW_DATA

Description of TIFF Tags

The following table lists the most frequently used TIFF tags. See enumIGTIFFTagIDs for a complete list of TIFF tags. For tags not listed in this table, see Non-Image Data Processing for information about how to find out whether a tag is read only or not.

Item Name	**Item Id**	**Read only**
"NewSubfileType"	254	TRUE
"SubfileType"	255	TRUE
"ImageWidth"	256	TRUE
"ImageLength"	257	TRUE
"BitsPerSample"	258	TRUE
"Compression"	259	TRUE
"PhotometricInterpretation"	262	TRUE
"Threshholding"	263	FALSE
"CellWidth"	264	TRUE
"CellLength"	265	TRUE
"FillOrder"	266	FALSE
"DocumentName"	269	FALSE
"ImageDescription"	270	FALSE
"Make"	271	FALSE
"Model"	272	FALSE
"StripOffsets"	273	TRUE
"Orientation"	274	FALSE[1]
"SamplesPerPixel"	277	TRUE
"RowsPerStrip"	278	TRUE
"StripByteCounts"	279	TRUE
"MinSampleValue"	280	FALSE
"MaxSampleValue"	281	FALSE
"XResolution"	282	FALSE
"YResolution"	283	FALSE
"PlanarConfiguration"	284	TRUE
"PageName"	285	FALSE
"XPosition"	286	FALSE
"YPosition"	287	FALSE
"FreeOffsets"	288	TRUE
"FreeByteCounts"	289	TRUE
"GrayResponseUnit"	290	FALSE
"GrayResponseCurve"	291	FALSE
"T4Options"	292	TRUE
"T6Options"	293	TRUE

"ResolutionUnit"	296	FALSE
"PageNumber"	297	TRUE
"TransferFunction"	301	FALSE
"Software"	305	FALSE
"DateTime"	306	FALSE
"Artist"	315	FALSE
"HostComputer"	316	FALSE
"Predictor"	317	TRUE
"WhitePoint"	318	FALSE
"PrimaryChromaticicities"	319	FALSE
"ColorMap"	320	TRUE
"HalftoneHints"	321	FALSE
"TileWidth"	322	TRUE
"TileLength"	323	TRUE
"TileOffsets"	324	TRUE
"TileByteCounts"	325	TRUE
"InkSet"	332	FALSE
"InkNames"	333	FALSE
"NumberOfInks"	334	FALSE
"DotRange"	336	FALSE
"TargetPrinter"	337	FALSE
"ExtraSamples"	338	TRUE
"SampleFormat"	339	FALSE
"SMinSampleValue"	340	FALSE
"SMaxSampleValue"	341	FALSE
"TransferRange"	342	FALSE
"JPEGTables"	347	TRUE
"JPEGProc"	512	TRUE
"JPEGInterchangeFormat"	513	TRUE
"JPEGInterchangeFormatLngth"	514	TRUE
"JPEGRestartInterval"	515	TRUE
"JPEGLosslessPredictors"	517	TRUE
"JPEGPointTransforms"	518	TRUE
"JPEGQTables"	519	TRUE
"JPEGDCTables"	520	TRUE
"JPEGACTables"	521	TRUE
"YCbCrCoefficients"	529	FALSE
"YCbCrSubSampling"	530	TRUE
"YCbCrPositioning"	531	FALSE
"ReferenceBlackWhite"	532	FALSE
"Copyright"	33432	FALSE
"IPTC/NAA"	33723	TRUE
"PhotoshopResources"	34377	FALSE

"ExifIFDPointer"	34665	TRUE
"GPSInfoIFDPointer"	34675	TRUE

1The tag is writable only with the IG_fltr_metad_update_file function.

TIFF Complex Data Tags

Some tags, which value is a complex data, can be passed in parsed form as following:

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
<TagName>	<TagId>	LEVEL_START	0	NULL	0	TRUE
"TagType"	<TagId>	VALUE_ITEM	UINT16	<TagType>	1	FALSE
"TagCount"	<TagId>	VALUE_ITEM	UINT32	<TagCount>	1	FALSE
<Tag Value Block>: one or more metadata items that identify the tag value						
<TagName>	<TagId>	LEVEL_END	0	NULL	0	TRUE

The <TagName> and <TagId> values are the tag name and identifier (available values see in the Table Photoshop Image Resource metadata structure, below). It is possible to use the tag identifier and value of nonstandard tag (user defined tag).

The <TagType> value is the standard TIFF tag type constant described in TIFF 6.0 specification.

Currently only TIFF tag with ID 34377, where Adobe Photoshop and some other TIFF writers save image recourses (IPTC data, resolution, some LUT etc.), is passed in this form. The <Tag Value Block> of these TIFF tag has structure described in the Table Photoshop Image Resource metadata structure below.

Photoshop Image Resource Metadata Structure

Name	**Id**	**Type**	**Value Type**	**Value**	**Value Length**	**Read Only**
"PhotoshopImageResource"	<Res. Id>	LEVEL_START	0	NULL	0	TRUE
"PhotoshopImageResourceSize"	<Res. Id>	VALUE_ITEM	UINT16		1	TRUE
Resource data						
"DATA"	<Res. Id>	VALUE_ITEM	RAW_DATA		Variable	FALSE
Or in case the data are parsed (currently it is happen only if the resource is IPTC data)						
"IPTC"	0x1C00	LEVEL_START	0	NULL	0	TRUE
...						
"IPTC"	0x1C00	LEVEL_END	0	NULL	0	TRUE
"PhotoshopImageResource"	<Res. Id>	LEVEL_END	0	NULL	0	TRUE

Where the <Res. Id> is the Adobe Photoshop image resource identifier (see Adobe Photoshop SDK).

IPTC PhotoshopImageResource identifier is 0x0404.

Callback Required for Writing TIFF Metadata Items

The value of TIFF metadata tags that are listed in the table below can be changed using LPAFT_IG_METAD_ITEM_SET_CB callback only:

Item Name	**Item Id**
"XResolution"	282
"YResolution"	283
"ResolutionUnit"	296
"PageNumber"	297
"Software"	305

"DateTime"	306
"Artist"	315
"FillOrder"	266
"DocumentName"	269

All the rest of the TIFF metadata tags values can be written using LPAFT_IG_METAD_ITEM_ADD_CB callback only.

1.2.6.8.2.8 XMP Non-Image Data Structure

ImageGear support XMP metadata in the following image file formats:

- TIFF: Read, Write
- EXIF-TIFF: Read, Write
- JFIF-JPEG: Read, Write
- EXIF-JPEG: Read, Write
- PSD: Read

In TIFF and EXIF-TIFF formats, XMP metadata is located in the main IFD.

In JFIF-JPEG and EXIF-JPEG formats, XMP metadata is located in one of the App1 segments. Note that EXIF metadata is also located in an App1 segment.

In PSD format, XMP metadata is located in the Photoshop Resources tree. If XMP.Parse global control parameter is TRUE, it is represented as XMP subtree. If XMP.Parse is FALSE, unparsed XMP metadata is located under PhotoshopImageResources subtree that has ID = 1060.

Brief information on XMP metadata levels is provided in the set of tables below:

- XMP Root Level
- XMP Schema Level
- XMP Namespace Level
- XMP Property Collection Level
- XMP Simple Property Level
- XMP Array Level
- XMP Array Items Level
- XMP Language Alternative Property Level
- XMP Structure Level
- XMP Structure Items Level

XMP Root Level

Name	Id	Type	Level Type	Value Type
<Schema namespace URI 1>	IGMDTAG_ID_XMP_DESCRIPTION	Tree	XMP Schema	N/A
<Schema namespace URI 2>	IGMDTAG_ID_XMP_DESCRIPTION	Tree	XMP Schema	N/A
...				

XMP Schema Level

Name	Id	Type	Level Type	Value Type
About	IGMDTAG_ID_XMP_ABOUT	Leaf	N/A	String
Namespace	IGMDTAG_ID_XMP_NAMESPACE	Tree	XMP Namespace	N/A
Properties	IGMDTAG_ID_XMP_NAMESPACE	Tree	XMP Property Collection	N/A

XMP Namespace Level

Name	Id	Type	Level Type	Value Type
Prefix	IGMDTAG_ID_XMP_PREFIX	Leaf	N/A	String
URI	IGMDTAG_ID_XMP_URI	Leaf	N/A	String

XMP Property Collection Level

Name	Id	Type	Level Type	Value Type
<Property 1	IGMDTAG_ID_XMP_PROPERTY	Tree	XMP Simple Property / XMP Array / XMP	N/A

name>			Structure	
<Property 2 name>	IGMDTAG_ID_XMP_PROPERTY	Tree	XMP Simple Property, XMP Array/Structure property	N/A
...				

XMP Simple Property Level

Name	**Id**	**Type**	**Level Type**	**Value Type**
Value	IGMDTAG_ID_XMP_PROPERTY_VALUE	Leaf	N/A	<Value type>
Qualifiers	IGMDTAG_ID_XMP_PROPERTY_QUALIFIERS	Tree	XMP Qualifiers collection	N/A

XMP Array Level

Name	**Id**	**Type**	**Level Type**	**Value Type**
Bag / Seq / Alt	IGMDTAG_ID_XMP_PROPERTY_BAG / IGMDTAG_ID_XMP_PROPERTY_SEQ / IGMDTAG_ID_XMP_PROPERTY_ALT	Tree	XMP Array items level	N/A

XMP Array Items Level

Name	**Id**	**Type**	**Level Type**	**Value Type**
Item	IGMDTAG_ID_XMP_PROPERTY	Tree	XMP Simple Property / XMP Language Alternative Property / XMP Structure	<N/A>
Item	IGMDTAG_ID_XMP_PROPERTY	Tree	XMP Simple Property / XMP Language Alternative Property / XMP Structure	<N/A>
...				

XMP Language Alternative Property Level

Name	**Id**	**Type**	**Level Type**	**Value Type**
Value	IGMDTAG_ID_XMP_PROPERTY_VALUE	Leaf	N/A	<Value type>
Lang	IGMDTAG_ID_XMP_PROPERTY_LANG	Leaf	N/A	String

XMP Structure Level

Name	**Id**	**Type**	**Level Type**	**Value Type**
Struct	IGMDTAG_ID_XMP_PROPERTY_STRUCT	Tree	XMP Structure items level	N/A

XMP Structure Items Level

Name	**Id**	**Type**	**Level Type**	**Value Type**
<Field 1 name>	IGMDTAG_ID_XMP_PROPERTY	Tree	XMP Simple Property	<N/A>
<Field 2 name>	IGMDTAG_ID_XMP_PROPERTY	Tree	XMP Simple Property	<N/A>
...				

1.2.7 Appendices/General Reference

This section provides referential information about ImageGear Professional.

1.2.7.1 Software License Agreement

PLEASE READ THE FOLLOWING SOFTWARE LICENSE AGREEMENT WHICH GOVERNS YOUR RIGHT TO USE OF THE TOOLKIT. YOU MUST ACCEPT THESE TERMS BEFORE YOU ARE ALLOWED TO INSTALL THE TOOLKIT. YOU EXPRESSLY AGREE THAT YOU HAVE THE AUTHORITY TO CONTRACTUALLY BIND THE ORGANIZATION AGREEING TO THESE TERMS.

BY CLICKING "I ACCEPT," OR INSTALLING TOOLKIT, OR PLACING TOOLKIT IN-USE, LICENSEE IS AGREEING TO BE BOUND BY THIS AGREEMENT.

1. GRANT OF AGREEMENT

This Accusoft Corporation, ("ACCUSOFT") Software License Agreement ("AGREEMENT") grants the organization contracting under this agreement as licensee ("LICENSEE") a limited, nontransferable, nonexclusive and non-assignable license to use the trial mode version of this ACCUSOFT Development Toolkit ("TOOLKIT") on a single computer for evaluation of fitness only and not for any commercial purpose; or to use a properly purchased and registered TOOLKIT, for development purposes only on a single computer, provided the TOOLKIT is IN-USE on only one computer at any time. (However additional TOOLKIT licenses may be purchased.) TOOLKIT is "IN-USE" on a computer when it becomes loaded by any means for any purpose into temporary memory (that is, including but not limited to RAM) or when it becomes copied or installed to less temporary storage by any means for any purpose (that is, including but not limited to hard disk, CD-ROM or other removable disk or tape, USB or other flash memory drive or card, or other local, networked, or cloud storage device) when it is accessible to that computer. The TOOLKIT is explicitly not to be used on a site-wide basis, via a server or other networked connection.

2. REDISTRIBUTION OF TOOLKIT RUNTIMES

ACCUSOFT does not grant LICENSEE any rights to deploy, license, sell, reproduce, copy, install, lease, timeshare, rent, or otherwise distribute or transfer TOOLKIT or any portion of TOOLKIT ("PORTION") except as provided in Section 1. GRANT OF AGREEMENT. For licensing information about any other distribution of TOOLKIT or PORTION, please visit our web site (www.accusoft.com/licensing.htm), or contact our sales staff. LICENSEE agrees to notify ACCUSOFT immediately of any violations or changes in status regarding LICENSEE's compliance with any term of this AGREEMENT.

In the event that ACCUSOFT grants LICENSEE in a written separate runtime license agreement ("RUNTIME AGREEMENT") a right to deploy, license, sell, reproduce, copy, install, lease, timeshare, rent, or otherwise distribute or transfer PORTIONS, the RUNTIME AGREEMENT will specify what PORTIONS may be distributed ("RUNTIME"). LICENSEE agrees to acknowledge and uphold the terms and conditions of this AGREEMENT as well as the terms of the RUNTIME AGREEMENT itself, which will be provided only in writing. In such event, LICENSEE may distribute RUNTIMES as part of the LICENSEE's software application or derivative works ("PRODUCT") upon additionally agreeing to the following:

a) LICENSEE understands and acknowledges that in order to receive any discounted pricing for RUNTIME distribution licensing fees based on the type of installation, it must either: 1) prepay for a number of RUNTIME licenses that is sufficient to qualify for ACCUSOFT's then-current published quantity discount, or 2) it must pay for the licenses in accordance with a written contract between LICENSEE and ACCUSOFT.

b) LICENSEE's PRODUCT shall not compete to any degree with the TOOLKIT. Such competitive PRODUCTS are defined as software development toolkits that include similar functionality as TOOLKIT and that are intended for use by software developers and/or system integrators.

c) LICENSEE's PRODUCT must be substantially greater in scope with greater functionality and features than those of the TOOLKIT.

d) LICENSEE will not use ACCUSOFT's name, logo, or trademarks to market PRODUCT without prior written approval of ACCUSOFT except LICENSEE will include a statement substantially similar to the following within PRODUCT documentation and about box: "Portions of this product contain imaging and other technology owned by Accusoft Corporation, Tampa, FL, (www.accusoft.com). ALL RIGHTS RESERVED." See Section 20. THIRD PARTY NOTICES for additional requirements.

e) LICENSEE agrees to only distribute the RUNTIMES. No license or other rights are granted to LICENSEE for any distribution of the TOOLKIT or PORTIONS including, but not limited to, documentation, source code, or the RUNTIME distribution unlock codes.

g) LICENSEE will only distribute the RUNTIMES on the hardware and operating system(s) for which the RUNTIMES are intended to be used according to the RUNTIME AGREEMENT.

If ANY of the terms of this AGREEMENT are not applicable to LICENSEE'S situation, or if any of the terms of this AGREEMENT cannot be complied with, or if LICENSEE needs modifications to this AGREEMENT or the license granted for any reason, LICENSEE must contact ACCUSOFT about obtaining an expanded license from ACCUSOFT (available by phone at: 813-875-7575, x321, by e-mail at: Sales@accusoft.com or by fax at: 813-875-7705).

This AGREEMENT grants rights to LICENSEE only for the TOOLKIT and does not convey any other rights of use or distribution to ACCUSOFT technology.

3. OWNERSHIP

LICENSEE acknowledges and agrees that ACCUSOFT owns all rights, title and interest in the TOOLKIT, in all forms, including without limitation any and all worldwide proprietary rights therein, including but not limited to trademarks, copyrights, patent rights, patent continuations, trade secrets and confidential information.

LICENSEE may not remove or alter the copyright notice from any copy of the TOOLKIT or any copy of the written materials, accompanying the TOOLKIT.

LICENSEE waives its right to contest any of ACCUSOFT's patents, trademarks, service marks, trade names, copyrights, and other intellectual property and proprietary rights in and to the TOOLKIT.

LICENSEE shall not use such trademarks, service marks, and trade names except where and as permitted under this AGREEMENT without receiving ACCUSOFT's prior written approval of such use. If such approval is granted, LICENSEE's right to use such trademarks, service marks, and trade names shall end upon the termination of this AGREEMENT.

4. RESTRICTIONS AND RESERVATIONS

All rights and licenses not expressly granted to LICENSEE are reserved to ACCUSOFT. LICENSEE is strictly prohibited from reproducing, copying, marketing, selling, distributing, licensing, sublicensing, leasing, timesharing or renting the TOOLKIT or PORTION, and LICENSEE is strictly prohibited from any use of the TOOLKIT or PORTION except as permitted by this AGREEMENT, and such actions are expressly prohibited. LICENSEE is strictly prohibited from incorporating or including the TOOLKIT or PORTION into or as part of any PRODUCT or service of LICENSEE except as provided by this AGREEMENT, regardless of the functionality of TOOLKIT (or lack thereof) within or as part of such PRODUCT or service of LICENSEE. LICENSEE shall not for any reason disassemble, decompile, decrypt or reverse engineer the TOOLKIT or PORTION or in any manner attempt to discover or reproduce the source code or any other copyrightable, proprietary, or trade secret aspect of the TOOLKIT or PORTION. Nor shall LICENSEE use the TOOLKIT or PORTION, directly or indirectly, in developing LICENSEE's own PRODUCT with, or including, similar functionality. LICENSEE shall not make any copies of the TOOLKIT or PORTION for any purpose whatsoever except as permitted by this AGREEMENT. Source code that is provided with TOOLKIT for sample or demonstration purposes may be used directly or indirectly in developing PRODUCT, however it may not be distributed in source form in whole or in part with or as part of PRODUCT.

5. WARRANTY DISCLAIMER

LICENSEE ACKNOWLEDGES AND AGREES THAT THE TOOLKIT IS PROVIDED "AS IS." ACCUSOFT DISCLAIMS ANY AND ALL REPRESENTATIONS AND WARRANTIES OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND AGAINST INFRINGEMENT.

6. LIMITATION OF LIABILITY

ACCUSOFT SHALL HAVE NO LIABILITY TO LICENSEE, LICENSEE AFFILIATES, SUBSIDIARIES, SHAREHOLDERS, OFFICERS, DIRECTORS, EMPLOYEES, REPRESENTATIVES OR ANY THIRD PARTY, WHETHER IN CONTRACT, TORT, NEGLIGENCE OR PRODUCTS LIABILITY, FOR ANY CLAIM, LOSS OR DAMAGE, INCLUDING BUT NOT LIMITED TO, LOST PROFITS, LOSS OF USE, BUSINESS INTERRUPTION, LOST DATA, LOST FILES, OR FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY KIND OR NATURE WHATSOEVER ARISING OUT OF OR IN CONNECTION WITH USE OF OR INABILITY TO USE THE TOOLKIT, OR THE PERFORMANCE OR OPERATION OF THE TOOLKIT, EVEN IF ACCUSOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. INDEMNIFICATION BY LICENSEE

LICENSEE SHALL INDEMNIFY, HOLD HARMLESS AND DEFEND ACCUSOFT FOR ANY LOSS, CLAIM, ACTION OR PROCEEDING THAT ARISES OR RESULTS FROM ANY ACTIONS OR OMISSIONS OF LICENSEE PERTAINING TO THE PRODUCT OR THE TOOLKIT AND FROM ANY ACTIONS OF LICENSEE THAT ARE IN VIOLATION OF THIS AGREEMENT.

8. TERM AND TERMINATION

Unless otherwise agreed to by the parties, this AGREEMENT shall become effective upon the earlier of LICENSEE's clicking of "I Accept" or LICENSEE'S installing or placing TOOLKIT IN_USE ("Effective Date") and shall continue in full force and effect until terminated in accordance with the terms set forth in this AGREEMENT.

Any material breach of this AGREEMENT shall automatically and immediately terminate this AGREEMENT. In the event that LICENSEE ceases to do business or is adjudged bankrupt or insolvent, ACCUSOFT may, at its sole option, terminate this AGREEMENT, by giving ten (10) Business Days written notice of such termination, which notice shall identify and describe the basis for such termination.

In the event of any termination of this AGREEMENT, any RUNTIME AGREEMENT is simultaneously terminated and LICENSEE shall stop using the TOOLKIT and PORTION, shall cease manufacturing the PRODUCT containing TOOLKIT or PORTION, and shall cease distributing PRODUCT containing TOOLKIT or PORTION. LICENSEE shall also require its resellers, OEMs, and other distribution channels (if any) to likewise stop manufacturing and distributing the PRODUCT containing TOOLKIT or PORTION. Within ten (10) Business Days thereafter, LICENSEE shall return or, at ACCUSOFT's option, destroy, the TOOLKIT and all PORTIONS, whether or not incorporated in or with the PRODUCT, that are within LICENSEE's possession, custody and control, and shall certify to ACCUSOFT in writing within ten (10) Business Days after that return or destruction that it has complied with the foregoing obligation.

All Sections except Section 1. GRANT OF LICENSE shall continue in full force and effect, notwithstanding any termination of this AGREEMENT.

9. LIQUIDATED DAMAGES

In the event LICENSEE (a) copies the TOOLKIT or PORTION except as permitted by this AGREEMENT, (b) uses the TOOLKIT or PORTION for any reason other than as permitted by this AGREEMENT, (c) installs or uses the TOOLKIT or PORTION on more than a single computer, or (d) otherwise violates or breaches this Agreement, LICENSEE agrees that ACCUSOFT is entitled to obtain as liquidated damages and not as a penalty the greater of the amount of (v) the published quantity one distribution price based upon the type of distribution, or (w) $99 per each user of each PRODUCT or service of LICENSEE in which the TOOLKIT or PORTION is included, copied, incorporated, embedded or accessible; (x) $100 per copy of TOOLKIT or PORTION; (y) $100 per copy of any PRODUCT in which TOOLKIT or PORTION is included, copied, incorporated, embedded or accessible; or (z) 3% of all revenues realized by LICENSEE pertaining to any PRODUCTS or services of LICENSEE in which TOOLKIT or PORTION is included, copied, incorporated, embedded or accessible. THE LICENSEE EXPRESSLY AGREES THAT THE FOREGOING LIQUIDATED DAMAGES ARE NOT A PENALTY.

10. CONFIDENTIALITY

LICENSEE acknowledges that the TOOLKIT contains ACCUSOFT know-how, confidential and trade secret information ("PROPRIETARY INFORMATION"). LICENSEE agrees: (a) to hold the PROPRIETARY INFORMATION in the strictest confidence, (b) not to, directly or indirectly, copy, reproduce, distribute, manufacture, duplicate, reveal, report, publish, disclose, cause to be disclosed, or otherwise transfer the PROPRIETARY INFORMATION to any third party, (c) not to make use of the PROPRIETARY INFORMATION other than as permitted by this AGREEMENT, and (d) to disclose the PROPRIETARY INFORMATION only to LICENSEE's representatives requiring such material for effective performance of this AGREEMENT and who have undertaken an obligation of confidentiality and limitation of use consistent with this AGREEMENT. This obligation shall continue as long as allowed under applicable law.

11. INJUNCTIVE RELIEF

LICENSEE agrees that any violation or threat of violation of this AGREEMENT will result in irreparable harm to ACCUSOFT for which damages would be an inadequate remedy. Therefore, in addition to its rights and remedies available at law (including but not limited to the recovery of damages for breach of this AGREEMENT), ACCUSOFT shall be entitled to immediate injunctive relief to prevent any violation of ACCUSOFT's copyright, trademark, trade secret rights regarding the TOOLKIT, or to prevent any violation of this AGREEMENT, including, but not limited to, unauthorized use, copying, distribution or disclosure of or regarding the TOOLKIT or PORTION, as well as any other equitable relief as the court may deem proper under the circumstances.

12. NO REDUCED PRICING

In any determination of ACCUSOFT's damages (whether liquidated damages or actual damages), or any determination of any licensing fees or royalties due ACCUSOFT under this AGREEMENT due to a breach by LICENSEE hereunder, LICENSEE shall not be entitled to any discounts (volume or otherwise) or reduced licensing fees or royalties. The foregoing sentence shall be applicable unless LICENSEE has negotiated and entered into a written, signed agreement with ACCUSOFT for such reduced or discounted licensing fees or royalties and paid ACCUSOFT such fees or royalties in advance of any: (a) distribution of the TOOLKIT or PORTION, (b) copying of the TOOLKIT or PORTION, or (c) incorporation or use of the TOOLKIT or PORTION in or pertaining to any PRODUCT or service of LICENSEE. Further, LICENSEE agrees that it shall not be entitled to reduced licensing fees or royalties when determining ACCUSOFT's damages due to any undertaking or activity by LICENSEE regarding the TOOLKIT or PORTION outside of or exceeding the scope of permission or other terms of this AGREEMENT, or LICENSEE's actions otherwise in violation of this AGREEMENT.

13. ATTORNEYS' FEES AND COSTS

In the event of any lawsuit or other proceeding brought as a result of any actual or alleged breach of this AGREEMENT, to enforce any provisions of this AGREEMENT, or to enforce any intellectual property or other rights in or pertaining to the TOOLKIT or PORTION, the prevailing party shall be entitled to an award of its reasonable attorneys’ fees and costs, including the costs of any expert witnesses, incurred at all levels of proceedings.

14. GOVERNING LAW

This AGREEMENT shall be construed, governed and enforced in accordance with the laws of the State of Florida, without regard to any conflicts of laws rules. Any action related to or arising out of this AGREEMENT will be filed only in the Florida courts and LICENSEE consents to the exclusive jurisdiction and venue of the state and federal courts located in Tampa, Florida.

15. SEVERABILITY

If any provision of this AGREEMENT is determined to be invalid by any court of final jurisdiction, then it shall be omitted and the remainder of the AGREEMENT shall continue to be binding and enforceable. In addition, the Court is hereby authorized to enforce any provision of the AGREEMENT that the Court otherwise deems unenforceable, to whatever lesser extent the Court deems reasonable and appropriate, rather than invalidating the entire provision. Without limiting the generality of the foregoing, LICENSEE expressly agrees that should LICENSEE be found to have breached the AGREEMENT, under no circumstances shall LICENSEE be entitled to any volume or other discount, or reduced licensing fee or royalty in the determination of ACCUSOFT's damages, or otherwise in the determination of any licensing fee or royalty owed to ACCUSOFT.

16. GOVERNMENT RIGHTS

The TOOLKIT and accompanying documentation have been developed at private expense and are sold commercially. They are provided under any U.S. government contracts or subcontracts with the most restricted and the most limited rights permitted by law and regulation. Whenever so permitted, the government and any intermediaries will obtain only those rights specified in ACCUSOFT's standard commercial license. Thus, the TOOLKIT referenced herein, and the documentation provided by Accusoft hereunder, which are provided to any agency of the U.S. Government or U.S. Government contractor or subcontractor at any tier shall be subject to the maximum restrictions on use as permitted by FAR 52.227-19 (June 1987) or DFARS 227.7202-3(a) (Jan. 1, 2000) or successor regulations. Manufacturer is Accusoft Corporation, 4001 N. Riverside Drive Tampa, FL 33603.

17. ENTIRE AGREEMENT

This AGREEMENT represents the entire understanding of the parties concerning the subject matter hereof and supersedes all prior communications and agreements, whether oral or written, relating to the subject matter of this AGREEMENT. Only a writing signed by the parties may modify this AGREEMENT. In the event of any modification in writing, of this AGREEMENT, including an expanded license agreement, all sections of this Agreement survive except Section 2. Limited License.

18. CONTACT US

Should you have any questions concerning this AGREEMENT, or if you desire to contact ACCUSOFT for any reason, please contact ACCUSOFT at 1-813-875-7575.

19. OTHER RESTRICTIONS

a) This AGREEMENT shall not be amended, altered, changed or modified in any way, unless agreed to in writing by both ACCUSOFT and LICENSEE. Such writing must be executed by a duly authorized representative of ACCUSOFT and a duly authorized representative of LICENSEE.

b) This AGREEMENT is not transferable or assignable by LICENSEE under any circumstances, without the prior written consent of ACCUSOFT. ACCUSOFT will not unreasonably withhold such consent. This AGREEMENT shall be binding upon, and is made for the benefit of, each party, its successors, and permitted assignees (if any). For the purposes of this AGREEMENT, any change in control of LICENSEE shall constitute an assignment or transfer of this AGREEMENT requiring prior written consent of ACCUSOFT. As used in this section, a change in control is defined as (i) any change in ownership of more than fifty percent (50%) of the voting interest in LICENSEE, whether by merger, purchase, foreclosure of a security interest or other transaction, or (ii) a sale of all or substantially all of the assets of LICENSEE.

c) The relationship established by this AGREEMENT between LICENSEE and ACCUSOFT shall be that of Licensee and Licensor. Nothing contained in this AGREEMENT shall be construed as creating a relationship of agency, joint venture or partnership between LICENSEE and ACCUSOFT. Neither party shall have any right whatsoever to incur any liabilities or obligations on behalf of the other party.

d) ACCUSOFT's failure to perform any term or condition of this AGREEMENT as a result of conditions beyond its control such as, but not limited to, war, strikes, fires, floods, acts of God, governmental restrictions, power failures, or damage or destruction of any network facilities or servers, shall not be deemed a breach of this AGREEMENT.

e) The headings provided in this AGREEMENT are for convenience and reference purposes only. In the event of a conflict between the terms and conditions listed in this AGREEMENT, and any attached Schedules or Appendices, the terms and conditions of this AGREEMENT shall govern.

f) A waiver of a breach, violation, or default under this AGREEMENT shall not be a waiver of any subsequent breach, violation or default. Failure of either party to enforce compliance with any term or condition of this AGREEMENT shall not constitute a waiver by the party of such term or condition.

g) All notices and communications shall be in writing and shall be deemed to have been duly given the earlier of when delivered or three (3) Business Days after mailing by certified mail, return receipt requested, postage prepaid, or by international delivery service, addressed to the parties at their respective addresses set forth on the Order Form or at such other addresses as the parties may designate by written notice in accordance with this section.

20. THIRD PARTY NOTICES

a) CAPTIVA ISIS TECHNOLOGY

ISIS functionality in the TOOLKIT is licensed to ACCUSOFT from Captiva Software, a division of EMC, Inc. The term "TOOLKIT" as defined in the Agreement includes technology from Captiva and related documentation, and any upgrades, modified versions, updates, additions, and copies thereof.

LIMITED WARRANTY

THE TERMS OF THIS AGREEMENT STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ACCUSOFT'S BREACH OF WARRANTY. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, CAPTIVA AND ITS SUPPLIERS MAKE NO WARRANTY, EXPRESS OR IMPLIED, AS TO MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR NON-

INFRINGEMENT. IN NO EVENT WILL CAPTIVA OR ITS SUPPLIERS BE LIABLE TO LICENSEE FOR ANY CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, EVEN IF A CAPTIVA REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY THIRD PARTY. Some states or jurisdictions do not allow the exclusion or limitation of incidental, consequential or special damages, or the exclusion of implied warranties, or limitations on how long an implied warranty may last, so the above limitations may not apply to LICENSEE. In such states/countries and to the extent permissible, any implied warranties are limited to thirty (30) days.

INTELLECTUAL PROPERTY

All ISIS technology is the intellectual property of Captiva and is protected under trademarks, registered trademarks, copyrights, and/or patents in the United States and/or other countries. ALL RIGHTS RESERVED.

b) ADOBE PDF TECHNOLOGY

Portions of the PDF functionality in the TOOLKIT include Adobe Technology ("Adobe") licensed to ACCUSOFT. The term "TOOLKIT" as defined in the Agreement includes technology from Adobe and related documentation, and any upgrades, modified versions, updates, additions, and copies thereof.

FONT LICENSE: If the TOOLKIT includes font software, LICENSEE may embed the font software, or outlines of the font software, into its Application to the extent that the font vendor copyright owner allows for such embedding. The fonts contained in this package may contain both Adobe and non-Adobe owned fonts. LICENSEE may fully embed any font owned by Adobe.

LIMITED WARRANTY

THE TERMS OF THIS AGREEMENT STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ACCUSOFT'S BREACH OF WARRANTY. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, ADOBE AND ITS SUPPLIERS MAKE NO WARRANTY, EXPRESS OR IMPLIED, AS TO MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT WILL ADOBE OR ITS SUPPLIERS BE LIABLE TO LICENSEE FOR ANY CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, EVEN IF AN ADOBE REPRESENTATIVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, OR FOR ANY CLAIM BY ANY THIRD PARTY. Some states or jurisdictions do not allow the exclusion or limitation of incidental, consequential or special damages, or the exclusion of implied warranties, or limitations on how long an implied warranty may last, so the above limitations may not apply to LICENSEE. In such states/countries and to the extent permissible, any implied warranties are limited to thirty (30) days.

INTELLECTUAL PROPERTY

The Adobe technology is the intellectual property of Adobe and is protected under trademarks, registered trademarks, copyrights, and/or patents in the United States and/or other countries. ALL RIGHTS RESERVED.

Rev. 20120723

1.2.7.2 Pixel Formats Supported by ImageGear Professional

ImageGear supports the following pixel formats for raster images:

Color Space	**Channel Depths (Bits per Channel)**
Indexed RGB: 1…8 bits per pixel	1…8
RGB	1…32
Grayscale	1…32
IHS	1…32
HLS	1…32
LAB	1…32
YIQ	1…32
CMY	1…32
CMYK	1…32
YCbCr	1…32
YUV	1…32
Extra	1…32
RGB + Alpha	1…32
Grayscale + Alpha	1…32
RGB + Pre-multiplied Alpha	1…32
Grayscale + Pre-multiplied Alpha	1…32
Indexed RGB + Extra	1…8
RGB + Extra	1…32
Grayscale + Extra	1…32
IHS + Extra	1…32
HLS + Extra	1…32
LAB + Extra	1…32
YIQ + Extra	1…32
CMY + Extra	1…32
CMYK + Extra	1…32
YCbCr + Extra	1…32
YUV + Extra	1…32
RGB + Alpha + Extra	1…32

Grayscale + Alpha + Extra	1…32
RGB + Pre-multiplied Alpha + Extra	1…32
Grayscale + Pre-multiplied Alpha + Extra	1…32

Note that the "Channel Depths" column lists bits per channel rather than bits per pixel. For example, for an RGB image, 1…32 bits per channel corresponds to 3…96 bits per pixel.

ImageGear allows image channels to have different depths. For example, these channel depths are allowed: RGB (5, 6, 5); CMYK + Extra (8, 8, 8, 16).

Not all of ImageGear functions support all pixel formats. Please see the API Reference for information on supported raster image formats for specific functions.

Please see the ImageGear Supported File Formats Reference for information on which pixel formats are supported for reading and writing for each image file format and compression method.

See Also

Understanding Bitmap Images

1.2.7.3 Understanding Bitmap Images

This chapter provides an introduction to the workings of the bitmap in the following sections:

1.2.7.3.1 Pixels

The pixel (or "picture element") is the basic building block of all images displayed by ImageGear. On a display screen, each pixel is a dot. In memory, the pixel is stored as a sequence of bits that determine what color the displayed pixel is. There can also be other data associated with the pixel.

1.2.7.3.2 Channels

An image is made up of one or more channels. For each pixel in the image, there are channel values for all of the image's channels. ImageGear supports the following channel types:

- Color channel - channel value describes a pixel's color. The number of color channels in an image is determined by the image's color space. For example, an RGB image has three color channels, one for each color component: red, green, and blue. CMYK has four color channels, and grayscale has only one.
- Alpha channel - channel value describes how much a pixel should be blended with another pixel during an alpha blending operation. In ImageGear, an image can have only one alpha channel, which can contain either non-premultiplied (as in PNG) or premultiplied (as in TIFF) alpha values. Also, only images with RGB or grayscale color channels can have alpha channels.
- Extra channel - extra data associated with the image on a per-pixel basis. For example, if an image being loaded contains multiple alpha channels, the first alpha channel will be loaded as the image's alpha channel, and the others will be loaded as extra channels. Extra channels are maintained in the image but do not contribute to display or alpha blending operations.

Each channel has a bit depth associated with it. This is the number of bits used to represent a value for the channel. This bit depth can be up to 32 bits on 32-bit platforms and up to 64 bits on 64-bit platforms. It is possible to have an image with multiple channels that differ in bit depth. For example, you could have an RGB image with 8 bits per color channel and a 1-bit alpha channel. Or you could have a 5-6-5 RGB image with 5 bits for red and blue channels and 6 bits for the green channel.

Previous versions of ImageGear stored alpha channels as separate, associated images. ImageGear now stores all channel values together for each pixel in an image.

1.2.7.3.3 Color Spaces

A color space in ImageGear describes the channels present in an image. The most important part of this description is the color channel configuration. This includes how many color channels there are and how these channels are used to describe colors, which is the usual informal concept of a color space. For example, in the RGB color space there are three channels (red, green, blue) whose values are combined to form colors. In the indexed color space there is one channel which consists of index values into an associated color palette.

In ImageGear, the concept of a color space is extended to also include information about other types of channels besides color, such as alpha and extra channels. An ImageGear color space ID is a bit field which can combine values from the enumIGColorSpaceIDs enumeration defined in accucnst.h.

For example,

- A simple RGB image would be:
  ```
  IG_COLOR_SPACE_ID_RGB
  ```
- A grayscale image with an alpha channel and no extra channels would be:
  ```
  IG_COLOR_SPACE_ID_Gy | IG_COLOR_SPACE_A
  ```
- An RGB image with a premultiplied alpha channel and three extra channels would be:
  ```
  IG_COLOR_SPACE_ID_RGB | IG_COLOR_SPACE_ID_P | IG_COLOR_SPACE_ID_Ex
  ```

1.2.7.3.4 24-bit RGB Images

24-bit RGB images are images in which each pixel is represented by three 8-bit quantities (thus, 24 bits total) specifying the intensities of red, green, and blue that form the color for the pixel. For example, a pixel that is to be displayed as brightest magenta-magenta being a color formed of equal intensities of red and blue, and no green-would be represented by three bytes having the respective values of 255, 0, 255 for the red, green, and blue intensities. In RGB representation the brightest white would be (255, 255, 255) and black would be (0, 0, 0).

1.2.7.3.5 1-bit Images

Images that contain only one bit per pixel can contain only two colors, since only two pixel values are possible: 0 and 1. The two colors for a 1-bit image are usually black and white, but if the image is to be displayed on an RGB device, the colors are determined by a table called a color palette. A color palette defines the color to be displayed for each possible pixel value. The following table demonstrates the color palette for a 1-bit image, which specifies black for pixel value 0 and white for pixel value 1:

	BLUE	GREEN	RED	Not Used
color 0	0	0	0	0
color 1	255	255	255	255

Each entry in a color palette is a 4-byte structure of type AT_RGBQUAD, in which the first three bytes are used to specify the intensities of blue, green, and red respectively, forming the color. The 4th byte is not used. (Note that the ordering in this structure is blue, green, red-not red, green, blue.) AT_RGBQUAD and all other structure types mentioned in this chapter are described in detail in the sections Core Component Data Types Reference and Core Component Structures Reference.

If instead you wanted to have pixel value 0 displayed as a medium yellow, and pixel value 1 as brightest red, you might use the following color palette (yellow is constructed of equal intensities of red and green, with no blue):

	BLUE	GREEN	RED	Not Used
color 0	0	128	128	0
color 1	0	0	255	0

1.2.7.3.6 4-bit and 8-bit Images

A 4-bit image is simply one in which each pixel is represented by 4 bits. Therefore, a 4-bit image can contain 16 (24) colors, each pixel having a numerical value between 0 and 15. The color palette for a 4-bit image will therefore normally have 16 entries (0 - 15.) As a 1-bit image might be called a 2-color image, a 4-bit image is also called a 16-color image.

In an 8-bit image each pixel occupies exactly one byte. This means each pixel has 256 (28) possible numerical values, from 0 to 255. Therefore, the color palette for an 8-bit image normally contains 256 entries, defining color 0 through color 255. 8-bit or 256-color images are very common because the availability of 256 unique colors provides adequate or even excellent color resolution for most purposes. Also, when operating upon an image in memory, such as when performing image analysis, transformations, or other image processing, operating on an 8-bit image is much faster than performing the same operations on a 24-bit image. And, of course, an 8-bit image uses only about one-third the memory or file storage space as a 24-bit image.

Because each pixel value of a 1, 4, or 8-bit color image is used as an index into a color palette, these images are sometimes called "indexed color" images. In this manual, 8-bit color images are sometimes referred to as "8i" images, to distinguish them from 8-bit gray level images, which are described below.

1.2.7.3.7 Grayscale Images

In RGB color representation, (255, 255, 255) results in the brightest white, and (0, 0, 0) results in black. Any time the three intensities are equal, no color is emphasized and the result is a shade of gray. For example (128, 128, 128) would be a medium gray, (240, 240, 240) would be a bright gray approaching white, and (16, 16, 16) would be a dark gray not far from black.

A grayscale image is one in which the color palette contains only grays, evenly graduated from black (0, 0, 0) for pixel value 0, to white (255,255,255) for the highest possible pixel value. This means that in an 8-bit gray level image, the blue, green, and red intensities for each palette entry are equal to the pixel value. The color palette for an 8-bit gray level image is illustrated below:

	BLUE	GREEN	RED	Not Used
color 0	0	0	0	0
color 1	1	1	1	0
color 2	2	2	2	0
color 3	3	3	3	0
...	...	...	...	...
color 254	254	254	254	0
color 255	255	255	255	0

Some ImageGear image processing operations cannot be performed on 8i (8-bit color) images, because some colors that occur as a result of the processing may not be present in the image's color palette. For such operations, if the image is 8-bit, it must be 8-bit gray level.

1.2.7.3.8 Color Values Used During Display

Although an image's color palette (or in the case of a 24-bit image, each pixel's 24-bit RGB value) normally determines the color to display at each pixel location, there are cases when this is not so. ImageGear maintains a set of Red, Green, and Blue "Look-Up Tables" (LUTs), which are used to determine whether the colors to actually display are different from those in the image's palette. The LUTs are modified, for example, if you instruct ImageGear to alter the brightness or contrast of an image. The LUTs can also be set directly by your application (meaning you can display an 8-bit grayscale image in any 256 colors of your choosing). See the Displaying Images, for further explanation of ImageGear's LUTs, including examples showing how to modify and use them.

The colors displayed for an image may also be modified due to constraints imposed by the display monitor being used. Some display monitors use a single 256-color hardware palette. ImageGear can reload this hardware palette each time an image is to be displayed, but since all images on the screen are being displayed using this one hardware palette, all other images on the screen at the same time will change to reflect the colors of this new palette. For such cases, you can instruct ImageGear as to which image or images are to have precedence in establishing the device's hardware palette. In addition, ImageGear can also inform you when the colors of an image may have changed due to the loading of another image's palette. This is discussed in more detail in the section Displaying Images.

1.2.7.3.9 Device-Independent/Device-Dependent Bitmaps

A bitmap image, also called a raster image, is an image held in the form of successive rows (called rasters) of pixel data. As already stated, ImageGear's bitmap images use 1, 4, 8, or 24 bits to represent each pixel. But besides these pixel bits, additional information is necessary to describe an image. For example, the width (number of pixels per row) and height (number of rows in the image), are required. For some images, a color palette may be required. Depending upon how the bitmap is to be used, additional information may be required or useful.

To hold bitmap images in memory, ImageGear uses two types of bitmaps, both widely used by many other software products and supported by development platforms such as Microsoft Windows. These are the Device-Independent Bitmap (DIB) and the Device-Dependent Bitmap (DDB). The DIB is by far the more widely used for in-memory operations. It has become the common denominator format because it is so simple: it describes just the image, without reference to characteristics of the device(s) upon which it may later be displayed. While ImageGear provides convenient ways to convert between DIBs and DDBs and to display DDBs, nearly all ImageGear operations are performed on DIBs exclusively.

1.2.7.3.9.1 Device-Independent Bitmaps (DIB)

A DIB consists of a header structure called a BITMAPINFOHEADER followed by the color palette, if one is present, followed by the bitmap (pixel) data:

BITMAPINFOHEADER
COLOR PALETTE (if any)
BITMAP DATA

The BITMAPINFOHEADER structure contains such information as the number of bits per pixel, number of pixels per row, and total number of rows in the image, as well as whether the bitmap data has been compressed for more efficient storage. Its form is shown below. Its fields are described in detail in the sections Core Component Data Types Reference and Core Component Structures Reference.

BITMAPINFOHEADER:	DWORD	biSize
	LONG	biWidth
	LONG	biHeight
	WORD	biPlanes
	WORD	biBitCount
	DWORD	biCompression
	DWORD	biSizeImage
	LONG	biXPelsPerMeter
	LONG	biYPelsPerMeter
	DWORD	biClrUsed
	DWORD	biClrImportant

A color palette is present in the DIB if the image is 1-bit, 4-bit, or 8-bit. The format of the color palette was described at the beginning of this chapter.

The format of the bitmap data is:

- For 1-bit images, the first (leftmost) pixel of a row is held in the Most Significant Bit (MSB) of the first byte of the row. Subsequent bits hold subsequent pixels, at a rate of 8 pixels per byte. The row is padded with zeroes as necessary to assure a multiple of 4 bytes length for each row.
- For 4-bit images, the first pixel of each row is in the 4 Most Significant Bits of the first byte of the row, and each succeeding 4 bits hold each succeeding pixel. As above, the row is zero-padded to a multiple of 4 bytes length.
- For 8-bit and 24-bit images, each pixel of a row occupies exactly 1 or 3 bytes respectively, and again each row is padded with zeroes to a multiple of 4 bytes length. Note that the colors in a 24-bit pixel in a standard DIB are ordered Blue-Green-Red (not Red-Green-Blue).

It should be noted that in a DIB, the first row of the bitmap data is the row to be displayed at the bottom of the image. For historical reasons, many file formats store their bitmap data top row first (this is because most devices to which bitmap data is sent display the rows top-to-bottom. Such devices, which include most CRT display monitors, are often called raster-scan devices.).

Whenever ImageGear loads a file of a format having top-to-bottom row ordering, it automatically reverses the order of the rows, assuring bottom-to-top row ordering for all DIBs, regardless of where the image originated. Keep in mind the ordering of a DIB is therefore "upside down" relative to the convention used in much display software that the top row of a display is row 0 and row numbers increase downward. However, it is important to note, ImageGear's pixel access functions (such as IG_DIB_pixel_set()) consider the coordinates 0,0 to refer to the upper left-hand corner of the bitmap data. As with an image shown on the screen, the x values will increase toward the right, and the y values will increase toward the bottom.

See Also:

ImageGear Architecture Diagram

1.2.7.3.9.2 Device-Dependent Bitmaps (DDBs)

Device-Dependent Bitmaps are bitmaps whose pixel data is organized for convenient dispatch to a particular device or group of devices, such as to a particular type of display monitor or printer. Normally you will not need to concern yourself with DDBs. ImageGear is designed to hold in-memory images in DIBs by default, and to process the image data of DIBs efficiently. But when the speed with which images are displayed is an important factor in your application, keeping large or often-displayed images in memory in DDB format may improve performance. For such cases, ImageGear provides several ways to create or import DDBs, and to display DDBs. Please refer to Using ImageGear.

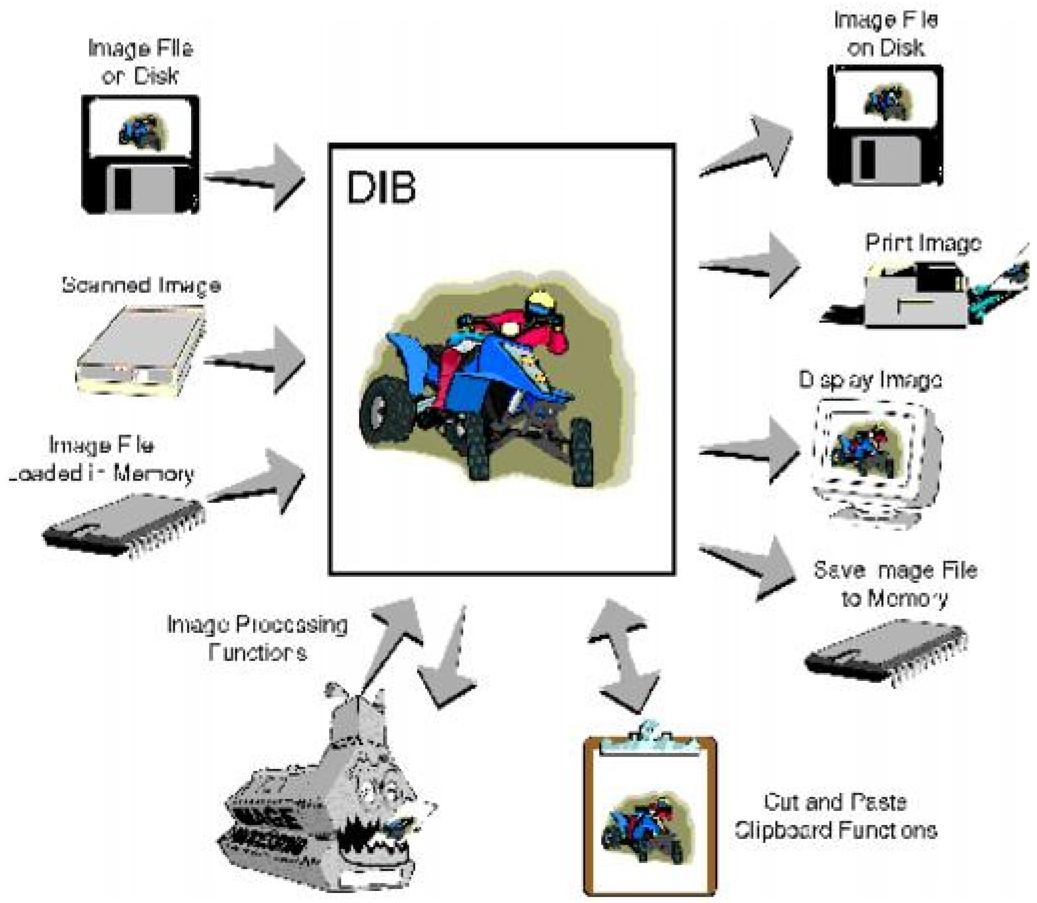
If your application is intensive in the use of in-memory DDB's, we suggest you refer to Microsoft Windows documentation of Device-Dependent Bitmaps and Device Contexts.

1.2.7.3.9.3 Vector Images

Bitmap images, sometimes also called raster images, have their data organized in horizontal rows of pixels for convenient dispatch to raster-scan devices. Devices and display software also exist that can directly display any line, if given an exact specification of the line. An example is a straight line defined by its two endpoints. Such a line is sometimes called a vector, and an image file whose data is specified exclusively in terms of vectors is called a vector image file.

ImageGear reads most popular formats that contains vector data. While doing so it performs so called "rasterization", i.e. conversion of vector data into raster representation. For some vector formats such conversion can only be possible for the certain platforms (WMF/EMF vector data can only be rasterized in Windows version of the product, and PICT vector data can only be rasterized in Mac version of the product). For some vector formats (DWF, DWG, DXF, HPGL,HPGL/2, DGN) ImageGear also allows you to control the projection of 3D vector data into 2D raster bitmap.

1.2.7.3.10 ImageGear Architecture Diagram



The diagram in figure 1 gives a conceptual view of the important role played by DIBs in ImageGear. The center image represents a DIB that results when an image is obtained from disk, memory, or a scanned file. Image processing and clipboard functions can be used to alter or merge the actual DIB bitmap data. To the right of the DIB are various routes an image may take after being manipulated by ImageGear: it may be saved to disk or to memory, it may be displayed, or it may be printed.

1.2.7.4 Function Error Return Codes

This Appendix delineates ImageGear error code names, numbers, and descriptions. All error and warning codes are listed in descending numerical order and divided into the groups specific to ImageGear functionality.

General Error Codes

IGE_SUCCESS	0	No errors - Success.
IGE_FAILURE	-1	General error - Failure.
IGE_NOT_LICENSED	-2	License error.
IGE_NOT_DONE_YET	-100	For internal reference of areas to which to return.
IGE_NOT_IMPLEMENTED	-200	For internal reference of areas to which to return.
IGE_NOT_SUPPORTED_BY_PLATFORM	-350	The last function used is not supported by this platform.
IGE_ERROR_COMPRESSION	-400	Compression error.
IGE_PARAMETER_HAS_INVALID_VALUE	-401	Incoming parameter is invalid.
IGE_INVALID_TYPE	-402	Incoming parameter's type is invalid.
IGE_OLD_CORE_CALL	-403	Internal failure. Contact Accusoft Technical Support.
IGE_BUFFER_HAS_INSUFFICIENT_SIZE	-404	Incoming buffer has insufficient size.
IGE_EXTENSION_NOT_LOADED	-500	The ImageGear extension was not present or couldn't be loaded.
IGE_FILTER_NOT_LOADED	-510	Requested format filter is not present.
IGE_INVALID_FILTER_OPERATION	-511	Requested operation is not supported by format filter.

IGE_FILTER_CTRL_INVALID_NAME	-512	The name of control parameter is invalid or not supported.
IGE_INVALID_FUNCTION_NAME	-550	The name of the component's function is invalid.
IGE_INVALID_FUNCTION_POINTER	-551	Invalid pointer to the component's function.
IGE_INVALID_COMPONENT_MODULE	-552	The module is not component or has wrong interface.
IGE_COMPONENT_ATTACH_FAILURE	-553	Failure to attach component.
IGE_INVALID_CONTROL_OPTION	-600	Invalid image control option ID.
IGE_INVALID_EXTENSION_MODULE	-700	The specified ImageGear extension file was not a valid extension file.
IGE_EXTENSION_INITIALIZATION_FAILED	-800	The specified ImageGear extension was unable to initialize.
IGE_FUNCTIONALITY_NOT_SUPPORTED	-900	The ImageGear functionality is not supported under this platform.
IGE_OUT_OF_MEMORY	-1000	No more global memory is available for allocation, reduced used resources.
IGE_EVAL_DLL_TIMEOUT_HAS_EXPIRED	-1003	The DLL is an Evaluation copy and as timed out - contact Accusoft to purchase a release copy.
IGE_INVALID_STANDARD_KERNEL	-1004	The kernel expected one of the predefined ones; yours could not be found.
IGE_INTERNAL_ERROR	-1005	An internal error has occurred, contact Accusoft technical support.
IGE_INVALID_RECTANGLE	-1007	Occurs when a rectangle's left >= right or top >= bottom.
IGE_NO_CLIPBOARD_IMAGE_AVAILABLE	-1008	No image is available for a clipboard paste.
IGE_CLIPBOARD_OPEN_FAILED	-1009	Could not open the clipboard.
IGE_SETCLIPBOARDDATA_FAILED	-1010	Could not put data into the clipboard.
IGE_COULD_NOT_GET_DDB_DIMENSIONS	-1011	GetObject() failed. Couldn't get the DDB's dimensions.
IGE_COULD_NOT_GET_DDB_BITS	-1012	GetDIBits() failed. Couldn't get the DDB's image data.
IGE_CREATE_BITMAP_FAILED	-1013	CreateBitmap() failed. Couldn't create a DDB.
IGE_COULD_DISPLAY_DDB	-1014	BitBlt() failed. Couldn't display the DDB.
IGE_INVALID_PATTERN_BITMAP	-1015	The DDB was > 1 bit per pixel or the width was > 8 or the height was > 8.
IGE_PASSWORD_INVALID	-1016	The Password is not recognized.

IGE_THUMBNAIL_NOT_PRESENT	-2000	Thumbnails are supported but none can be found in this image file.
IGE_THUMBNAIL_READ_ERROR	-2001	A read error occurred while reading a thumbnail.
IGE_THUMBNAIL_NOT_SUPPORTED	-2002	Thumbnails are not supported by this format.
IGE_PAGE_NOT_PRESENT	-2005	The requested image page does not exist in the file.
IGE_PAGE_INVALID	-2006	The page number provided is outside of the range of valid pages for this file.
IGE_PAGE_COULD_NOT_BE_READ	-2007	The page number could not be determined.
IGE_CANT_DETECT_FORMAT	-2010	The format of the file can not be determined.
IGE_FILE_CANT_BE_OPENED	-2030	An attempt to open a file failed; it may not exist in the provided path.
IGE_FILE_CANT_BE_CREATED	-2031	An attempt to create a file failed; it may already exist in the provided path.
IGE_FILE_CANT_BE_CLOSED	-2032	An attempt to close a file failed.
IGE_FILE_TO_SMALL_TO_BE_BMFH	-2033	The file is too small to be a valid BMFH.
IGE_FILE_IS_NOT_BMP	-2034	The BMFH Magic number is invalid.
IGE_FILE_TO_SMALL_TO_BE_BMIH	-2035	The file is too small to be a valid BMIH.
IGE_BMP_IS_COMPRESSED	-2040	The BMP file is in compressed (RLE) format.
IGE_FILE_SIZE_WRITE_ERROR	-2041	Could not write file size field to BMP.
IGE_CANT_READ_PALETTE	-2050	Can't read palette.
IGE_CANT_READ_PIXELS	-2051	Can't read pixel data.
IGE_CANT_READ_HEADER	-2052	Can't read header.
IGE_INVALID_FILE_TYPE	-2060	Invalid file type.
IGE_INVALID_HEADER	-2061	Invalid file header.
IGE_CANT_WRITE_PALETTE	-2070	Can't write palette.
IGE_CANT_WRITE_PIXELS	-	Can't write pixel data.

	2071	
IGE_CANT_WRITE_HEADER	-2072	Can't write header.
IGE_FORMAT_NOT_DETECTABLE	-2073	Save format cannot be detected from file extension used.
IGE_INVALID_COMPRESSION	-2080	Invalid compression.
IGE_INSTANCE_FAILURE	-2090	Instance failure.
IGE_INSTANCE_CLEANUP_ERROR	-2091	Instance cleanup error.
IGE_CANT_SETUP_DIB_HEADER	-2095	Can't set up DIB header.
IGE_CANT_READ_FILE	-2100	Can't read file.
IGE_INVALID_IMAGE_FORMAT	-2110	The image file is invalid as the expected format.
IGE_FILE_FORMAT_IS_READONLY	-2111	The image file is read only and cannot be written to.
IGE_INVALID_BITCOUNT_FOR_FORMAT	-2112	The bitcount found is not supported by this format.
IGE_INTERRUPTED_BY_USER	-2113	Status bar callback returned FALSE.
IGE_NO_BITMAP_REGION	-2390	No bitmap region.
IGE_BAD_FILE_FORMAT	-2391	Format is not correct.
IGE_EPS_NO_PREVIEW	-2392	EPS file has no screen preview image to load.
IGE_CANT_WRITE_FILE	-2393	File can't be saved in the specified format.
IGE_NO_BITMAP_FOUND	-2394	WPG, WMF, etc. No raster image exists in file.
IGE_PALETTE_FILE_TYPE_INVALID	-2395	IG_PALETTE_ value is not known.
IGE_PALETTE_FILE_WRITE_ERROR	-2396	Error writing to a palette file.
IGE_PALETTE_FILE_READ_ERROR	-2397	Error reading from a palette file.
IGE_PALETTE_FILE_NOT_DETECTED	-2398	The file is not a valid palette file.
IGE_PALETTE_FILE_INVALID_HALO_PAL	-2399	Detected Dr. Halo Palette file is invalid.

IGE_G4_PREMATURE_EOF_AT_SCAN_LINE	-2400	Group 4 premature EOF.
IGE_G4_PREMATURE_EOL_AT_SCAN_LINE	-2401	Group 4 premature EOL.
IGE_G4_BAD_2D_CODE_AT_SCAN_LINE	-2402	Group 4 invalid 2D code.
IGE_G4_BAD_DECODING_STATE_AT_SCAN_LINE	-2403	Group 4 bad decoding state.
IGE_G3_PREMATURE_EOF_AT_SCAN_LINE	-2410	Group 3 premature EOF.
IGE_G3_BAD_1D_CODE_AT_SCAN_LINE	-2411	Group 3 bad 1D code.
IGE_G3_PREMATURE_EOL_AT_SCAN_LINE	-2412	Group 3 premature EOL.
IGE_BITDEPTH_NOTSUPPORTED	-2413	This Bit-Depth is not supported for this write format.
IGE_DIRECTORY_CREATE_ERROR	-2414	Unable to create Destination Directory.
IGE_LOG_FILE_CREATE_ERROR	-2417	Unable to create Batch Log File.
IGE_NAME_CONV_NOT_SUPPORTED	-2416	Batch Naming configuration not supported.
IGE_IMNET_INVALID_WIDTH	-2418	Invalid width for IMNET.
IGE_PJPEG_INVALID_SCAN_CONFIGURATION	-2420	Invalid configuration of scans for progressive JPEG write.
IGE_PJPEG_INVALID_SCAN_COUNT	-2421	Invalid number of scans for progressive JPEG write.
IGE_JPG_UNRECOGNIZED	-2422	Unrecognized JPEG marker encountered.
IGE_JPG_INVALID_QTABLE_ID	-2423	Invalid quantization table descriptor.
IGE_JPG_INVALID_QTABLE_PRECISION	-2424	Invalid quantization table precision.
IGE_JPG_INVALID_HUFFMAN_ID	-2425	Invalid Huffman table descriptor.
IGE_JPG_INVALID_HUFFMAN_TABLE	-2426	Invalid Huffman table.
IGE_PJPEG_NOT_SUPPORTED	-2427	Progressive JPEG feature is not supported.
IGE_OPERATION_IS_NOT_ALLOWED	-2432	This operation is not allowed.
IGE_PROC_INVAL_FOR_RUNS_DIB	-	This function can not be used on DIBs in the Runs

	2433	format - convert first IG_IP_convert_runs_to_DIB.
IGE_CAN_NOT_OPEN_TEMP_FILE	-2434	The temporary file need for this function could not be opened/created.
IGE_ALLOC_SELECTOR_FAILED	-2435	AllocSelector() failed, couldn't get an entry into the Global Descriptor able.
IGE_LOAD_FUNCTION_GET_FAILED	-2436	Was not able to initialize the filer load function.
IGE_PNG_CHUNK_WRITE_FAILED	-2438	Failed to write the correct number of bytes for png chunk.
IGE_PNG_WRITE_FAILED	-2439	Failed to write png data.
IGE_PNG_CHUNK_READ_FAILED	-2440	Failed to READ the correct number of bytes for png chunk.
IGE_PNG_READ_FAILED	-2441	Failed to READ png data.
IGE_PNG_NO_IDAT_CHUNK	-2442	Failed to READ a mandatory IDAT Chunk.
IGE_NOT_SUPPORTED_COMP	-2443	Compression is not supported at this time.
IGE_UNDEFNIED_COLOR_SPACE_ID	-2444	Color space ID is not defined.
IGE_DIB_RES_UNITS_NOT_SUPPORTED	-2445	DIB resolution units are not supported.
IGE_FILTER_CANT_GET_INFOFUNC	-2446	Failed to get filter's info function.
IGE_FILTER_UNKNOWN_FORMAT	-2447	Unknown file format.
IGE_X_NULL_DISPLAY	-2448	X display specified is NULL. ASCII filter is unable to draw text.

TIFF Filter Specific Errors

IGE_INVALID_TAG	-2450	Tag Read did not contain correct number of bytes.
IGE_INVALID_IFD	-2451	IFD Read did not contain correct number of bytes.
IGE_IFD_PROC_FAILURE	-2452	Invalid IFD information was detected.
IGE_SEEK_FAILURE	-2453	IOS position seek failed.
IGE_INVALID_BYTE_ORDER	-2454	Byte order flag was not Intel or Motorola.
IGE_CANT_READ_TAG_DATA	-2455	Was unable to read all TAG information.

IGE_INVALID_BITS_PER_SAMPLE	-2456	Bits per sample tag was invalid.
IGE_INVALID_COLOR_MAP	-2457	Color Map was found to be invalid.
IGE_INVALID_PHOTOMETRIC	-2458	Photometric tag value was found to be invalid.
IGE_INVALID_REQ_INFO	-2459	Required information was not supplied.
IGE_COMP_NOT_SUPPORTED	-2460	Compression is not supported at this time.
IGE_RASTER_FEED_ERROR	-2461	Error feeding raster data to the DIB.
IGE_IMAGE_DATA_READ_ERROR	-2462	Was unable to read all image data requested.
IGE_HEADER_WRITE_FAILED	-2463	Header write failed.
IGE_DIB_GET_FAILURE	-2464	Was unable to retrieve the DIB information.
IGE_CANT_REALLOC_MEM	-2465	Was not able to reallocate memory.
IGE_IFD_WRITE_ERROR	-2466	Was not able to write IFD info to the IOS.
IGE_TAG_WRITE_ERROR	-2467	Was not able to write TAG info to the IOS.
IGE_IMAGE_DATA_WRITE_ERROR	-2468	Was not able to write IMAGE data to the IOS.
IGE_PLANAR_CONFIG_ERROR	-2469	Planar Config detected is unsupported.
IGE_RASTER_TO_LONG	-2470	One raster lines exceeds the max number of bytes.
IGE_LZW_ERROR	-2471	Error occurred in LZW decode.
IGE_INVALID_IMG_DEM	-2472	Image Dimension was invalid.
IGE_BAD_DATA_TYPE	-2473	Data type detected is not valid.
IGE_PAGE_COUNT_FAILURE	-2474	Count not count the number of pages in the file.
IGE_CORRUPTED_FILE	-2475	Data in file was not what was expected and could not be interpreted.
IGE_INVALID_STRIP_BYTE_CNT	-2476	Strip byte count was zero and could not be estimated.
IGE_INVALID_COMP_BIT_DEPTH	-	Bit depth is invalid for this compression scheme.

	2477	
IGE_REPAGE_FAILED	-2478	Unable to write new page numbers while re-paging file.
IGE_PRIV_TAG_TYPE_INVALID	-2479	Private user tag had an invalid type.
IGE_LZW_EXTENSION_NOT_LOADED	-2480	LZW Extension has not been loaded.
IGE_TILE_NOT_PRESENT	-2481	Tile is not present.
IGE_RASTER_WRITE_FAILURE	-2482	Unable to write Raster to Output Device (Full Device).
IGE_IMBEDDED_IMAGE_FAILURE	-2483	Failure occurred while reading a file format imbedded in another.
IGE_ABIC_EXTENSION_NOT_LOADED	-2484	ABIC Extension has not been loaded.
IGE_JBIG_EXTENSION_NOT_LOADED	-2485	JBIG Extension has not been loaded.
IGE_JBG_IMG_CNTRL_NOT_FOUND	-2486	JBIG Extension Image Control not found for save.
IGE_CLP_INVALID_FORMAT_ID	-2500	Windows clipboard file contains an unsupported Format ID at this page.
IGE_ICA_COMP_NOT_SUPPORTED	-2510	MO:DCAIOCA Compression is not supported at this time.
IGE_ICA_IBM_MMR_COMP_ERROR	-2520	Error in IBM MMR IOCAMO:DCA compression.
IGE_TIF_INVALID_CLASS_F_IMAGE	-2530	Error writing TIF class F format.
IGE_JBIG_STREAM_OPEN_FAILURE	-2540	JBIG support library returned an error in return code.
IGE_CANNT_OPEN_FTP_FILE	-2550	Can't open FTP file.
IGE_CANNT_OPEN_HTTP_FILE	-2560	Can't open HTTP file.
IGE_CANNT_OPEN_GOPHER_FILE	-2570	Can't open Gopher file.
IGE_CANNT_OPEN_TEMPORARY_FILE	-2580	Can't open temporary file.
IGE_CANNT_OPEN_INTERNET_CONNECTION	-2590	Can't open internet connection.
IGE_CANNT_OPEN_INTERNET_SESSION	-2600	Can't open internet session.
IGE_END_OF_IMAGE	-2610	End of image.

Format Filter Warning Codes

IGW_FILTER_DECODING_FAILURE	-2650	Failed to decode rasters.
IGW_INVALID_COMPRESSION_NONCRITICAL	-2651	Invalid compression of non-critical data, e.g. text.

Sync Error Codes

IGE_IMAGE_IS_LOCKED	-2900	Image is locked.
IGE_CANT_LOCK_IMAGE	-2901	Can't lock image.
IGE_CANT_UNLOCK_IMAGE	-2902	Can't unlock image.
IGE_CANT_LOCK_DATA	-2903	Can't lock data.
IGE_CANT_UNLOCK_DATA	-2904	Can't unlock data.

Image Processing Error Codes

IGE_WRONG_DIB_BIT_COUNT	-3000	The DIB has bit with the wrong bit count for this routine.
IGE_LOCK_FAILED	-3010	Memory required for this routine could not be locked; most likely running out of memory resources.
IGE_ALLOC_FAILED	-3020	Memory required for this routine could not be allocated; free up some resources and try again.
IGE_FREE_FAILED	-3030	An internal memory free has failed, usually a bad handle, or corrupted system.
IGE_BAD_KERN_TYPE	-3040	Bad kernel type.
IGE_AI_HANDLES_USED_UP	-3050	The maximum number of Accusoft handles has been used - no more left. Free up some and try again.
IGE_AI_HANDLE_INVALID	-3060	This routine requires an Accusoft handle. The handle passed in was not allocated by Accusoft.
IGE_DIBS_ARE_INCOMPATIBLE	-3070	The images are not compatible for this function, either dimension, bit count, or both.
IGE_INVALID_SIGMA	-3080	Invalid sigma.
IGE_DIB_DIMENSIONS_NOT_EQUAL	-3090	The images must be the same dimensions.
IGE_DIB_BIT_COUNTS_NOT_EQUAL	-3100	The images must have the same bit count.

IGE_DIB_HAS_NO_PALETTE	-3101	The image passed in does not have a palette and this routine requires one.
IGE_ROI_WRONG_TYPE	-3110	Region of interest is wrong type.
IGE_REQUIRES_CONVEX_ROI	-3120	This function requires a convex ROI. The one passed in must be convex.
IGE_INVALID_RAMP_DIRECTION	-3130	The contrast ramps direction is not supported.
IGE_INVALID_LUT_ARITH_FUNC	-3140	The LUT_ARITH_FUNC is not a valid function number; check the constant.
IGE_INVALID_KERN_MOTION_EXTENT	-3150	Invalid kernel motion extent.
IGE_INVALID_NOISE_TYPE	-3160	Invalid noise type.
IGE_INVALID_KERN_NORMALIZER	-3170	Invalid kernel normalizer.
IGE_INVALID_SIGMA	-3180	Invalid sigma.
IGE_INVALID_SKEW_POINTS	-3190	A valid line could not be drawn through the two point provided. Y1==Y2.
IGE_TILE_IS_LARGER_THAN_IMAGE	-3200	The tile image must be the same size or smaller in both dimensions than the original source.
IGE_COLOR_SPACE_INVALID	-3210	Invalid type of color space.
IGE_DIB_POINTER_IS_NULL	-3220	The DIB pointer about to be used is NULL (invalid).
IGE_PROC_INVAL_FOR_BIT_COUNT	-3300	This function can not be used on images with this one's bit count.
IGE_PROC_INVAL_FOR_PALETTE_IMG	-3310	This function can not be used on 8-bit color images - try to promote to 24-bit.
IGE_PARAMETER_OUT_OF_LIMITS	-3320	A parameter is out of its legal range.
IGE_INVALID_POINTER	-3330	A pointer was detected to be NULL.
IGE_INVALID_ENCRYPT_MODE	-3340	The selected Encryption Method is invalid.
IGE_PASSWORD_LENGTH_INVALID	-3350	Password must be at least 1-byte long (should be at least 5 bytes).
IGE_PROC_REQUIRE_8BIT_GRAYSCALE	-3360	This function can be used on 8-bit grayscale images only.
IGE_PROC_REQUIRE_8BIT_GRAYSCALE_1CHANNEL	-3361	This function can only be used on 8-bit grayscale images with one channel.
IGE_PROC_REQUIRE_8_16_32BIT_GRAYSCALE_1CHANNEL	-	This function can only be used on 8, 16, or 32-

	3362	bit grayscale images with one channel.
IGE_INVALID_RESOLUTION_UNIT	-3370	The units of the image resolution are not supported.
IGE_POINTER_IS_NULL	-3380	Pointer passed to an IP function is NULL, but it would point at object.
IGE_INVALID_BIT_MASK	-3390	The red, green and blue components of the bit mask overlap.
IGE_DIB_DIMENSIONS_ARE_INVALID	-3400	Either height or width of the DIB is wrong.
IGE_PROC_INVAL_FOR_8BIT_INDEXED	-3410	Proc does not work on the 8i images.
IGE_RASTER_LINE_INVALID	-3420	The given raster line is invalid, it should be between 0 and the height of the image -1.
IGE_INVALID_CLIPING_RECT	-3421	Invalid clipping rect.
IGE_INVALID_ALPHA_CHANNEL	-3422	It is possible that width or height not equal to original image.
IGE_INVALID_RESOLUTION_VALUE	-3423	Resolution value is incorrect.
IGE_INVALID_RANK_TYPE	-3424	Specified type of rank in the rank filter is invalid.
IGE_RASTER_DIB_REQUIRED	-3425	Raster IG page required for this operation.

Disk File Access Error Codes

IGE_FILE_CANT_OPEN	-3440	Cannot open file.
IGE_FILE_CANT_SAVE	-3441	Cannot save file.
IGE_FILE_CANT_DELETE	-3442	Cannot delete file.
IGE_FILE_INVALID_FILENAME	-3443	Invalid file name.
IGE_FILE_INVALID_PATH	-3450	Invalid path.

Batch Conversion CB Error Codes

IGE_BATCH_WRONG_CB_TYPE	-3500	Wrong CB type passed to IG_batch_CB_register.

Auto Detect Error Codes

IGE_FILE_IS_SYSTEM_FILE	-3600	The image file passed in is really one of the following system files and not an image.

IGE_FILE_IS_EXE	-3601	File is a EXE, DLL, DRV, FNT, OCX, or 386.
IGE_FILE_IS_ZIP	-3602	File is a PKZIP file.
IGE_FILE_IS_DOC	-3603	File is a Microsoft DOC file.
IGE_FILE_IS_HLP	-3604	File is a Microsoft system Help file.
IGE_FILE_IS_UNSUPPORTED_FILE	-3605	File format is not supported.

Component Related Error Codes

IGE_CANT_ATTACH_COMP	-3700	The specified component can't be attached.
IGE_COMPONENT_NOT_ATTACHED	-3701	The component isn't attached.
IGE_FUNCTION_DOESNT_EXIST	-3702	The function doesn't exist in the specified component.

General Warning Codes

IGW_GENERAL_WARNING	-4750	See detailed description in text field of warning.
IGW_LAST_ITEM_REACHED	-4751	Iterator has reached the final item.

Display Error Codes

IGE_CANT_OPEN_PARAMETER_MUTEX	-4800	Cannot start critical session.
IGE_CANT_OPEN_FORMAT_STORAGE_MUTEX	-4801	Cannot start critical session.
IGE_FAIL_TO_ALLOC_PAMETER_GROUP	-4805	Cannot allocate parameter group.
IGE_THREAD_ALREADY_ASSOCIATED	-4810	Thread is already associated.
IGE_INVALID_OBJECT	-4811	Invalid object.
IGE_GROUP_IS_NOT_EXIST	-4817	Group does not exist.
IGE_FAIL_TO_ALLOC_MUTEX	-4820	Cannot allocate critical session data.
IGE_INVALID_FORMAT_METHOD_ID	-4825	Invalid format method ID.
IGE_INVALID_FORMAT_HEADER	-	Invalid format header.

	4830	
IGE_INVALID_FORMAT_BIT_COUNT	-4835	Invalid format bit count.
IGE_FORMAT_OPERATION_IS_NOT_SUPPORTED	-4840	Format operation is not supported.
IGE_INVALID_DISPLAY_PARAMETER	-4850	Invalid display parameter.
IGE_INVALID_DEVICE_CONTEXT	-4860	Invalid device context.
IGE_INVALID_IMAGE_HANDLE	-4870	Invalid image handle.
IGE_INVALID_PARAMETER	-4880	Invalid parameter.
IGE_INVALID_PRM_OPERATION	-4890	Invalid operation.
IGE_FAIL_TO_DRAW_IMAGE	-4910	Cannot draw image.

AVI Warning Codes

IGW_AVI_PARTIAL_BAD_FRAMES	-4900	Partial bad frames warning.

PS Warning Codes

IGW_PS_UNKNOWN_PAGES_NUMBER	-4950	Unknown page number warning.

PDF Read Function Error Codes

IGE_PDFREAD_PSERROR	-5300	PostScript error.
IGE_PDFREAD_GENERAL	-5301	General error.

PS2 TEXT Function Error Codes

IGE_PS2TEXT_ERROR	-5350	PS2TEXT error.

Multipage Error Codes

IGE_MP_INVALID_HANDLE	-5400	Invalid multi-page image handle.
IGE_MP_IMVALID_PAGE_NUMBER	-5401	Invalid page number.
IGE_MP_CANT_OPEN_MUTEX	-5402	Synchronization error.

IGE_MP_ASSOCIATE_FAILURE	-5403	Multipage image open failure.
IGE_MP_IMAGE_NOT_ASSOCIATED	-5404	Multipage image is not opened.

Multipage Warning Codes

IGW_MP_INVALID_PAGE_NUMBER	-5420	Invalid page number.

PDF/PostScript Component Error Codes

PostScript General

IGE_PS_FAILURE	-5501	General PostScript job error.
IGE_PS_NOT_INITIALIZED	-5502	PostScript engine is not initialized.
IGE_PS_EMPTY_JOB	-5503	Empty PostScript job.
IGE_PS_POSTSCRIPT_ERROR	-5504	The file contains a PostScript error.

PDF General

IGE_PDF_FAILURE	-5531	General PDF error.
IGE_PDF_NOT_INITIALIZED	-5532	PDF engine is not initialized.
IGE_PDF_CANT_READ_DATA	-5533	Cannot read PDF stream.
IGE_PDF_CANT_WRITE_DATA	-5534	Cannot write PDF stream.
IGE_PDF_CANT_CONVERT_DATA	-5535	Rasterization failed.
IGE_PDF_INVALID_PARAMETER	-5536	Invalid parameter.
IGE_PDF_INVALID_COMPRESSION	-5537	Invalid compression.
IGE_PDF_INVALID_BITCOUNT	-5538	Invalid bit count.
IGE_PDF_CANT_EXTRACT_TEXT	-5539	Cannot extract text.
IGE_PDF_CANT_CREATE_OBJECT	-5540	Cannot create object.
IGE_PDF_PRINTING_FAILED	-5541	Printing failed.

PDF Document

IGE_PDF_DOC_INVALID	-5601	Invalid document object.
IGE_PDF_DOC_CANT_CREATE	-5602	Cannot create document.
IGE_PDF_DOC_CANT_OPEN	-5603	Cannot open document.
IGE_PDF_DOC_CANT_SAVE	-5604	Cannot save document.
IGE_PDF_DOC_NOT_AUTHORIZED	-5605	This operation is not permitted.

PDF Page

IGE_PDF_PAGE_INVALID	-5701	Invalid page object.
IGE_PDF_PAGE_CANT_CREATE	-5702	Cannot create page.
IGE_PDF_PAGE_CANT_OPEN	-5703	Cannot open page.
IGE_PDF_PAGE_CANT_SAVE	-5704	Cannot save page.
IGE_PDF_PAGE_CANT_INSERT	-5705	Cannot insert page.
IGE_PDF_PAGE_CANT_DELETE	-5706	Cannot delete page.

PDF Objects

IGE_PDF_OBJECT_INVALID	-5801	Invalid PDF object.
IGE_PDF_CONTENT_INVALID	-5802	Invalid content object.
IGE_PDF_CONTENT_CANT_SET	-5803	Cannot set content.
IGE_PDF_ELEMENT_INVALID	-5804	Invalid element.
IGE_PDF_COLORSPACE_INVALID	-5805	Invalid color space object.
IGE_PDE_OBJECT_INVALID	-5806	Invalid PDE object.
IGE_PDF_STREAM_INVALID	-5807	Invalid stream object.

GUI Function Error Codes

IGE_REGISTER_CLASS_FAILED	-6000	Microsoft Windows function: RegisterClass() failed.

IGE_CREATE_WINDOW_FAILED	-6010	Microsoft Windows function: CreateWindow() failed.
IGE_WINDOW_NOT_ASSOCATED	-6020	An attempt was made to disassociate a window that was never associated.
IGE_INVALID_WINDOW	-6030	An invalid window handle was passed as one of the parameters to the function.
IGE_UNREGISTER_CLASS_FAILED	-6040	Microsoft Windows function: UnregisterClass() failed.
IGE_GUI_NO_FORMATS_CTL	-6500	The control is not associated with a valid IGFormatsCtl control.
IGE_GUI_NO_DISPLAY_CTL	-6501	The control is not associated with a valid IGDisplayCtl control.
IGE_GUI_NO_PROCESSING_CTL	-6502	The control is not associated with a valid IGProcessingCtl control.
IGE_GUI_NO_PAGE_DISPLAY	-6503	Invalid IGPageDisplay object.
IGE_GUI_NO_PAGE	-6504	Invalid IGPage object.
IGE_GUI_PAGE_DISPLAY_CREATE_OBJ_FAILED	-6505	The CreatePageDIsplay() call failed.
IGE_GUI_PAGE_DISPLAY_SET_OBJ_FAILED	-6506	Could not set the PageDisplay object for this image's PageView Control.
IGE_GUI_LAYOUT_GET_OBJ_FAILED	-6507	Could not retrieve the DIsplay Layout object for this image display.
IGE_GUI_LAYOUT_CHANGE_FAILED	-6508	Could not modify the properties of the Display Layout object for this image display.
IGE_GUI_DISPLAY_ZOOM_GET_OBJ_FAILED	-6509	Could not retrieve the Display Zoom object for this image display.
IGE_GUI_DISPLAY_ZOOM_SET_OBJ_FAILED	-6510	Could not set the Display Zoom object for this image display.
IGE_GUI_DISPLAY_ZOOM_CHANGE_FAILED	-6511	Could not modify the properties of the Display Zoom object for this image display.
IGE_GUI_PAGE_VIEW_ENABLE_FAILED	-6512	Could not enable the PageView Control.
IGE_GUI_PAGE_VIEW_UPDATE_FAILED	-6513	Could not update the PageView Control.
IGE_GUI_GET_DLG_CTL_FAILED	-6514	Could not retrieve the dialog control object.
IGE_GUI_GET_HWND	-6515	Could not retrieve the underlying HWND for this object.
IGE_GUI_UNKNOWN_FILE_FORMAT	-6516	Unknown file format.
IGE_GUI_INTERNAL_ERROR	-	Internal GUI error.

	-6517	
IGE_GUI_SLIDER_SETUP_FAILED	-6518	Failed to setup magnifier's on-source sliding area.
IGE_GUI_WINDOWING_FAILED	-6519	Failed to attach to the source view control window.
IGE_GUI_SOURCE_VIEW_UNAVAILABLE	-6520	SourceView property must be setup before starting mouse tracking.
IGE_GUI_DESTINATION_VIEW_UNAVAILABLE	-6521	DestinationView property must be setup in stationary mode before starting mouse tracking.
IGE_GUI_GET_DEVICE_RECT_FAILED	-6522	Failed to retrieve device rectangle from SourceView's page display.

VBX/OCX Level Error Codes

IGE_VBX_INVALID_FUNCTION_NUM	-7000	Invalid VBX function number.
IGE_ROTATE_ENUMERATED_VALUES_NOT_USED	-7010	The enumerated values for rotate were not used.
IGE_GUIWINDOW_TYPE_INVALID	-7020	GUIWindow type invalid.

OS2 Error Codes

IGE_UNREGISTERED_HAB	-8000	HAB is not registered.

NRA Error Codes

IGE_INVALID_REGION_DATA	-9000	Invalid region.
IGE_INVALID_PARAMETER_FOR_PROCESSING_WITH_REGION	-9001	Operation cannot be done using region.

XMP Metadata Error Codes

IGE_XMP_METADATA_ERROR	-9780	An XMP metadata error has occurred.

Last Error and Warning Codes

IGE_LAST_ERROR_NUMBER	-9999	Last error code number. Application defined error codes must be negative and less than this number. Note that it may change in a major release, so make sure to use a constant rather than a hard-coded number.
IGW_LAST_WARNING_NUMBER	-9999	Last warning code number. Application defined error and warning codes must be negative and less than this number. Note that it may change in a major release, so make sure to use a constant rather than a hard-coded number.

1.2.7.5 ImageGear Global Control Parameters

The following table provides information about ImageGear Professional Global Control Parameters:

Global Control Parameter	**Type**	**Default Value**	**Available Values**	**Description**
COMM.PATH	AM_TID_LP | AM_TID_CHAR	<Path to the folder where ImageGear Core module is located>	Any string	Specifies the path to the folder containing ImageGear component modules. See IG_comm_comp_attach.
CPM.CMYK_PROFILE_PATH	AM_TID_LP | AM_TID_CHAR	<Path to the default CMYK profile; located in the same directory as ImageGear Core module.>	Any string	Specifies the path to the default CMYK color profile. See Using Color Profile Manager.
CPM.ENABLE_PROFILES	AM_TID_AT_BOOL	FALSE	FALSE, TRUE	Enables or disables the color profile manager. See Using Color Profile Manager.
CPM.ENABLE_RENDERING_INTENTS	AM_TID_AT_BOOL	FALSE	FALSE, TRUE	Enables or disables the usage of rendering intents in color profiles. By default, ImageGear uses Relative Colorimetric rendering intent. See Using Color Profile Manager.
CPM.RGB_PROFILE_PATH	AM_TID_LP | AM_TID_CHAR	<Path to the default RGB profile; located in the same directory as ImageGear Core module.>	Any string	Specifies the path to the default RGB color profile. See Using Color Profile Manager.
DIB.FILE_MAPPING.FLUSH_SIZE	AM_TID_INT	200	Any non-negative integer	Specifies the maximum size of a memory block in a DIB that can be processed without flushing the memory mapped file, in megabytes. Only used when "DIB.FILE_MAPPING.THRESHOLD" is greater than zero. Does not affect 1-bit images. See Working with Gigabyte-Sized Images.
DIB.FILE_MAPPING.PATH	AM_TID_LP | AM_TID_CHAR	<Empty string. ImageGear uses the system temporary folder for memory mapped files.>	Any string	Specifies the path to a folder where memory mapped files will be stored. Only used when "DIB.FILE_MAPPING.THRESHOLD" is greater than zero. Does not affect 1-bit images. See Working with Gigabyte-Sized Images.
DIB.FILE_MAPPING.THRESHOLD	AM_TID_INT	0	Any non-negative integer	Specifies minimum DIB size, in megabytes, for which the memory mapped file shall be used. Zero means that the use of memory mapped files is disabled. Does not affect 1-bit images. See Working with Gigabyte-Sized Images.
DIB.PIX_ACCESS_USE_LEGACY_MODE	AM_TID_AT_LMODE	IG_PIX_ACCESS_MODE_LEGACY	IG_PIX_ACCESS_MODE_LEGACY = 0, IG_PIX_ACCESS_MODE_NEW = 1	Switches between “legacy” (14.4 and earlier) and “new” pixel access modes. See Pixel Access Modes.
FLTR.METADATA_FORMAT	AM_TID_LP | AM_TID_CHAR	“binary”	“binary”, “text”	Switches between binary and text representations for accessing metadata values. See Non-Image Data Format.
IO.BUFFER_SIZE	AM_TID_DWORD	262144	Any non-negative integer	Specifies the size of the internal buffer used for image reading. Set to 0 to disable bufferization. See Internal Stream Bufferization.
PDF.HOST_FONT_PATH	AM_TID_LP | AM_TID_CHAR	<Parameter is not set. ImageGear uses the system fonts directory.>	Any string	Specifies path to the system font directory. See IG_PDF_initialize.
PDF.PDF_RESOURCE_PATH	AM_TID_LP | AM_TID_CHAR	<Parameter is not set. ImageGear looks for PDF resources in <COMM.PATH>\Resources\PDF>	Any string	Specifies the path to the PDF resources directory. See IG_PDF_initialize.
PDF.PS_RESOURCE_PATH	AM_TID_LP | AM_TID_CHAR	<Parameter is not set. ImageGear looks for PDF resources in <COMM.PATH>\Resources\PS>	Any string	Specifies the path to the PostScript resources directory. See IG_PDF_initialize.
PDF.TMP_PATH	AM_TID_LP | AM_TID_CHAR	<Parameter is not set. ImageGear selects PDF/PS	Any string	Specifies the path to the temporary directory for PDF

		temporary directory according to system settings.>		component. See IG_PDF_initialize. This parameter specifies the location where temp objects such as scratch PDF files, scratch PS files, and PS font cache are created. This parameter should be set before the PDF component is initialized with IG_PDF_initialize.
PRINT.RESOLUTION	AM_TID_INT	300	Positive integer	Specifies both vertical and horizontal resolution in dots per inch during image printing. A higher value indicates a higher quality image to be printed.
XMP.Parse	AM_TID_AT_BOOL	TRUE	FALSE, TRUE	Enables or disables the parsing of XMP stream when reading and writing metadata from/to image files. See Working with XMP Metadata.

1.2.7.6 Glossary

This glossary contains terminology used by Accusoft in both its software products and its documentation. Since these terms come from many different disciplines and because many of the terms have different meanings in each discipline, each glossary entry is followed by the name of the field from which the definition is taken.

For a detailed description, refer to the references listed in the Bibliography. For definitions of ImageGear licensing terminology, refer to ImageGear Licensing and Deployment Kit Terminology.

To find terms which start with a numeral (0-9), look under its spelling, for example, the term "8-bit gray level" can be found as "eight bit gray level."

absolute coordinates (imaging)

Absolute coordinates refer to a common origin, for example, the upper left corner of a display screen. This is the opposite of relative coordinates.

ACCUAPI (Accusoft)

Accusoft Application Program Interface. See API (software).

additive primary colors (imaging)

Red, Green, Blue - the 3 colors used to create all other colors when direct, or transmitted light is used (as in a video monitor). They are called additive primaries, because when these three colors are superimposed they produce white.

anti-aliasing (imaging)

A method of filling in data that is missing due to under-sampling. In imaging, this usually involves the process of removing jagged edges by interpolating values in-between pixels of contrast. These methods are most often used to remove or reduce the stair-stepping artifact found in digital high contrast images.

AOI (Image Processing)

Area Of Interest. An area of interest is a rectangle within an image defined as two points within the image. An AOI can be written as (x1,y1)-(x2,y2). All AOIs are parallel with the image's axes. See ROI (Accusoft image processing).

API (software)

Application Programmer's Interface. The set of routines that make up a library or toolkit. Some times called a binding.

aspect ratio (imaging)

The proportion of an image's size given in terms of the horizontal length verses the vertical height. An aspect ratio of 4:3 indicates that the image is 4/3 times as wide as it is high.

Bezier curve (graphics)

A curve created from endpoints and two or more control points that serve as positions for the shape of the curve. Originated by P. Bezier (~1962) for the use in car body descriptions.

bit block transfer

A raster operation that moves a block of bits representing a portion of an image or scene from one location in the frame buffer to another. Usually written as "bit blt".

bin (image processing)

See histogram (imaging).

Bit Block Transfer (Windows)

An optimized movement of a large block of computer memory from one location to another. Used for moving images or sub-images to and from areas of computer memory.

bit blt

bit_block_transfer.

bitmap (imaging)

An image is a bitmap if it contains a value for each of its pixels. This is the opposite of vector images where a small set of values generate an object.

bit plane (imaging)

A hypothetical 2-D plane containing a single bit of memory for each pixel in a image. If each 8-bit pixel is thought of as a stack of 8 coins, and an image as many rows and columns of these stacked coins then the 3rd bit plane would be the plane consisting of the 3rd coin from each stack.

bounding rectangle (geometry)

The smallest rectangle that fits around a given object. In imaging, the rectangle is usually rotationally restricted to be parallel to both image axes.

.BMP (file format extension)

Format originator: Microsoft Corporation

16011 NE 36th Way, Box 97917

Redmond, WA 98073

Call-back function (software)

A function that is passed to another function as a parameter. The function receiving the call-back function can call this function. This is used to change the behavior of a given routine without knowing beforehand what it is expected to do.

Cartesian coordinates (imaging)

A 2-dimensional equally spaced grid iron that uniquely assigns every point in the plane, (one and only one), co-ordinate pair; (x, y). In imaging, each point is usually referred to as a pixel and the x and y values take on integer values. Most images use the top-left as the (0,0), or origin. See coordinates.

Chroma-key (image processing)

An image blending function that replaces pixels of a specified hue range with pixels from a second image. This is often referred to the weatherman effect because most weather forecasters use a solid blue or green background to make it look as if they are standing in front of a huge weather map. It is important to remember that it is the hue that is used in the blending function and not the intensity or saturation.

C.I.E (color imaging)

Commission Internationale de l'Eclairage. (International Commission of Illumination). A standards organization which provides specifications for the description of device independent color.

clipboard (Windows)

The clipboard is a windows data structure used to exchanged data between applications. It is a common area where applications place data and others can access it. These operations are usually referred to as Cut (place data in) and Paste (take data out).

closing (image processing)

See MPEG (image compression).

CMY & CMYK

Cyan, Magenta, Yellow, (K) black. Computer monitors are additive, but color printers are subtractive. Instead of combining light from monitor phosphors, printers coat paper with colored pigment that removes specific colors from the illumination light.

CMY is the subtractive color model that corresponds to the additive RGB model. Cyan, magenta, and yellow are the color complements of red, green, and blue. Due to the difficulties of manufacturing pigments that produce black when mixed together, a separate black ink is often used and is referred to as K (`B' is already used for blue).

color map (imaging)

See Look-Up-Table (computer hardware).

color model (imaging)

See color space (imaging).

color space (imaging)

A mathematical coordinate system (space) for assigning numerical values to colors. There are many ways to define such spaces, each with its own benefits and problems.

See Also:

- CMY & CMYK
- HIS (color imaging)
- HLS (color imaging)
- HSV (color imaging)
- RGB (imaging)
- YIQ (color imaging)

compression (imaging)

An image processing method for saving valuable disk and memory space by reducing the amount of space required to save a digital image. The graphics data is rewritten allowing it to be represented by a smaller set of data. Do not confuse this with encoding. See lossless (image compression) and lossy (image compression).

compression ratio (imaging)

The ratio of a file's uncompressed size over its compressed size.

concave (geometry)

A 2-dimensional blob, for example, a region of interest (ROI), where at least one tangent is drawn that touches the blob at two different locations, and there is a point on the tangent between the two contacts that does not touch the blob.

In simpler words, if a rubber band could be snugly wrapped around a concave blob there would be places where the rubber band lifts off and does not touch the blob. Concave is the opposite of convex.

convex (geometry)

A 2-dimensional blob, for example, a region of interest (ROI), where every tangent that can be drawn touches the blob at a continuous stretch of the blob's surface with no gaps.

In simpler words, if a rubber band could be snugly wrapped around a convex blob there would be no places where the rubber band lifts off and is not touching the blob. Convex is the opposite of concave.

convolution (image processing)

An image processing operation that is used to spatially filter an image. A convolution is defined by a kernel that is a small matrix of fixed numbers. The size of the kernel, the numbers within it, and a single normalizer value define the operation that is applied to the image. The kernel is applied to the image by placing the kernel over the image to be convolved and sliding it around to center it over every pixel in the original image.

At each placement the numbers (pixel values) from the original image are multiplied by the kernel number that is currently aligned above it. The sum of all these products is tabulated and divided by the kernel's normalizer. This result is placed into the new image at the position of the kernel's center. The kernel is translated to the next pixel position and the process repeats until all image pixels have been processed.

As an example, a 3x3 kernel holding all `1's with a normalizer of 1/9 performs a neighborhood averaging operation. Each pixel in the new image is the average of its 9 neighbors from the original.

coordinates

A pair of numbers that represent a specific location in a two-dimensional plane, for example, an image or on a map.

See Also:

- absolute coordinates (imaging)

- device coordinates (imaging)
- Cartesian coordinates (imaging)
- polar coordinates (imaging)
- relative coordinates
- screen coordinates (imaging)
- world coordinates

crop (Imaging)

An image processing method for removing the region near the edge of the image, but keeping the central area.

.DCX (file format extension)

Format originator: Intel

DDB (Windows)

Device-Dependent Bitmap. A Window image specification that depends on the capabilities of a specific graphics display controller. Since a DDB is matched to the current graphics controller, it is fast and easy to display since large blocks of memory need only be copied to the controller.

See Also:

DIB (Windows).

decompression (imaging)

When an image or other digital data set is compressed and stored, it is not usable until it is decompressed into it original form.

device coordinates (imaging)

The co-ordinates of the coordinate system that describe the physical units that defines the computer screen.

device dependent (software)

Software written to work on a specific set of hardware platforms. Since these routines make use of physical device attributes, they may behave differently on other devices, although they will most often not work on other devices.

See Also:

- device independent (software)
- DIB (Windows)

device driver (software)

A set of low-level software routines that work with and control a specific hardware device. The names and functions are often standardized across many similar devices. This allows higher level software to use the hardware as a generic device. This frees the higher-level software from dealing with the particulars of specific devices and allows devices to be interchanged.

device independent (software)

Software or data structures that are designed to work with or on a wide set of hardware platforms.

See Also:

- device independent (software)
- DIB (Windows)

DIB (Windows)

Device-Independent Bitmap is a Windows-defined image format specification. It is called device-independent because of its straightforward, common-denominator, format. It has all the information that a basic digital image needs and is laid out in a simple specification. Its simplicity makes it an ideal format for holding images that need to be shared by several programs.

See Also:

- DDB (Windows)
- The book *Programming Windows* by Charles Petzold

dilation (image processing)

See MPEG (image compression)

dithering (imaging)

The method of using neighborhoods of display pixels to represent one image intensity or color. This method allows low-intensity resolution display devices to simulate higher resolution images. For example, a binary laser printer can use block patterns to display grayscale images.

See Also:

halftone (imaging)

DLL (Microsoft Windows)

Dynamic Linked Library. A compiled and linked collection of computer functions that are not directly bound to an executable. These libraries are linked at run-time by Windows. Since Windows is in charge of managing (loading, linking, and removing) the DLLs, they are available to all executables currently running. Each executable links to a shared DLL saving memory by avoiding redundant functions from co-existing. DLLs allow a new level of modularity by providing a means to modify and update executables without re-linking. Just copy a new version of the DLL to the correct disk directory.

DPI (printing)

Dots Per Inch. The number of printer dots that can be printed in one inch. The printer's resolution is defined by the number of dots per inch: lower resolution = less dots per inch, higher resolution = more dots per inch.

edge (image processing)

In an image, an edge is a region of contrast or color change. Edges are useful in machine vision since optical edges often mark the boundary of physical objects.

edge detection (image processing)

A method that isolates and locates an optical edge in a digital image.

edge map (image processing)

An edge map is the output of an image-processing filter that transforms an image into an image where intensity represents a change in the contrast (optical edge) of the original image.

(eight) 8-bit image (digital imaging)

An image where each pixel has 8-bits of information. An 8-bit pixel contains one of 256 possible values. There are two common types of 8-bit images: grayscale and indexed color.

In a grayscale image, each pixel takes one of 256 shades of gray and the shades are linearly distributed from 0 (black) to 256 (white). An 8-bit grayscale image does not require a palette but may have one.

An indexed color image is always a palette image. Each pixel is used as an index to the palette. These images can have up to 256 different colors. This includes hues as well as shades. Indexed 8-bit images are good for low color resolution images that do not need processing. They are 3 times smaller than full-color RGB images, but because the pixel values are not linear, many image-processing algorithms cannot work with them. They must be promoted to 24-bit for image processing.

8-bit gray level (Accusoft term)

This indicates 8-bit grayscale. 8-bit gray level is used to distinguish between 8-bit indexed color (8i) and 8 bit grayscale. An 8-bit gray level DIB image is one where each pixel in the bitmap is unchanged by its palette when displayed. Each palette entry is the same as its index.

8i (Accusoft term)

This indicates 8-bit indexed color. 8i is used throughout this manual to distinguish between 8-bit grayscale (8-bit gray level) and 8-bit indexed color. An 8-bit indexed color DIB is one where each 8-bit pixel value in the bitmap is used as an index to the palette.

The palette dictates which RGB color the pixel displays. These images are compact ways of storing color images. However they are difficult to process because the bytes that make up the pixel can no longer be ordered with any certainty.

Encoding

The format for storing uncompressed data (binary, ASCII, etc.), how it is packed (e.g. 4-bit pixels may be packed at a rate of two pixels per byte), and the unique set of symbols used to represent the range of data items.

.EPS (file format extension)

Format originator: Adobe Systems, Inc.

1585 Charleston Road

Mountain View, CA 94039

equalize (image processing)

An image-processing algorithm that redistributes the frequency of image pixel values allowing equal representation for any given continuous range of values. In an ideal world, an equalized image has the same number of pixels in the range from 10-20 as it does from 200-210. However, since digital images have quantized intensity values, the range totals are rarely identical but usually close.

erosion (image processing)

See MPEG (image compression)

file format (software)

A specification for holding computer data in a disk file. The format dictates what information is present in the file and how it is organized.

filter (image processing)

An image-processing filter is a transform that removes a specified quantity from an image. For instance a spatial filter removes high, medium or low spatial frequencies from an image.

(four) 4 bit image (digital imaging)

An image file format that allows 4-bits per pixel. This image can contain up to 16 (24) different colors or levels of gray.

frame (imaging)

A single picture, usually taken from a collection of images for example, a movie or video stream.

frame buffer (imaging hardware)

A computer peripheral that stores and sometimes manipulates digital images.

frame processes (image processing)

Image-processing algorithms that operate on a single image.

fx (imaging)

See special effects (image processing)

gain & level (imaging)

Gain and level are image-processing terms that correspond to the brightness and contrast control on a television. The gain is the "contrast", and the level is the "brightness." By changing the level, the entire range of pixel values are linearly shifted brighter or darker. Gain on the other hand linearly stretches or shrinks the intensity range, altering the contrast.

gamma correction (imaging)

A non-linear function that is used to correct the inherent non-linearities of cameras and monitors. The intensity of the luminescent phosphor on the raster display is non-linear. Gamma correction is an adjustment to the pixel intensity values that make up for this inherent non-linearity.

geometric transform (image processing)

A class of image processing transforms that alter the location of pixels. This class includes rotates and warps.

.GIF (file format extension)

Name: Graphics Interchange File Format

Format originator: CompuServe Inc.

500 Arlington Center Blvd.

Columbus, OH 43220

This format uses the LZW compression created by Unisys. It is the same as the LZW compression used in the TIFF file format, except that the bytes are reversed and the string table is upside-down.

All GIF files have a palette. Some GIF files can be interlaced - the raster lines can appear as every 4 lines, then every 8 lines, then every other line. This is due to GIF files usually being received from a modem.

GUI

Graphical User Interface. A computer-user interface that uses graphical objects and a mouse for user interaction, for example Microsoft Windows.

graphics library (software)

A collection of software routines that work on digital images. These collections usually contain routines for drawing various graphical objects, for example, lines, circles, and rectangles.

gray level (imaging)

A shade of gray assigned to a pixel. The shades are usually positive integer values taken from the grayscale. In an 8-bit image a gray level can have a value from 0 to 255.

grayscale (imaging)

A range of gray levels. Zero is usually black and higher numbers indicate brighter pixels.

group III Fax (Imaging compression)

A CCITT standard for transmission of facsimile data. It compresses black and white images using a combination of differential, run length and Huffman coding.

halftone (imaging)

The reproduction of a continuous-tone image on a device that does not directly support continuous output. This is done by displaying or printing a pattern of small dots that simulate the desired output color or intensity. These methods are used extensively in magazines and newspapers.

handle (software)

A handle references a data object. A handle is a type of pointer but it usually contains, internally, more information about the referenced object.

histogram (imaging)

A tabulation of pixel value populations displayed as a bar chart where the x-axis represents all the possible pixel values and the y-axis is the total image count of each given pixel value. A histogram counts how many pixels in the image have a given intensity value or range of values.

Each histogram intensity value or range of values is called a bin. Each bin contains a positive number that represents the number of pixels in the image that fall within the bin's range. A typical 8-bit grayscale histogram contains 256 bins. Each bin has a range of a single intensity value. Bin 0 contains the number of pixels in the image that have a grayscale value of 0 or black; bin 255 contains the number of white (255) pixels. When the collection of bins are sorted (0-255) and charted, the graph displays the intensity distributions of all the images pixels.

HLS (color imaging)

Hue Saturation, and Lightness. A method that describes any color as a triplet of real values. The hue represents the color or wavelength of the color. It is sometimes called tone and is commonly known as color. The hue is taken from the standard color wheel and is calibrated in degrees.

Saturation is the depth of the color. It states how gray the color is. It is a real valued parameter from 0.0 to 1.0 with 0.0 indicating full gray and 1.0 representing pure hue.

Lightness determines how black or white a color is. It ranges from 0.0 to 1.0 but with 0.0 representing black and 1.0 white. A lightness of 0.5 is a pure hue.

HSV (color imaging)

Hue, Saturation, and Value.

Huffman coding (image compression)

A method of encoding symbols that varies the length of the code in proportion to its information content. Groups of pixels that appear frequently in a image are coded with fewer bits than those of lower occurrence.

HIS (color imaging)

Intensity,Hue, and Saturation.

image format (image storage)

There are many digital image formats. Some of these are: TIFF, DIB, GIF, and JPEG. The image format specification dictates which image information is present and how it is organized in memory. Many formats support various sub-formats or `flavors'.

image processing

The general term "image processing" refers to a computer discipline wherein digital images are the main data object. This type of processing can be broken down into several sub-categories: compression, image enhancement, image filtering, image distortion, image display and coloring, and image editing.

See Also:

machine vision

indexed color image (imaging)

An image where each pixel value is used as an index to a palette for interpretation before the pixel is displayed. These images contain a palette that is initialized specifically for a given image. The pixel values are usually 8-bit and the palette 24-bit (8-red, 8-green, and 8-blue).

See Also:

(eight) 8-bit image (digital imaging)

invert intensity (image processing)

An image processing operation where each pixel is subtracted from the maximum pixel value allowed. This produces a photographic negative of the original. For an 8-bit image the inverse function is:

invert(pix) = 255-pix;

For an 8-bit RGB image the function is:

invert(Rpix) = 255-Rpix;

invert(Gpix) = 255-Gpix;

invert(Bpix) = 255-Bpix;

"jaggies" (imaging)

A term used to describe the visual appearance of lines and shapes in raster pictures that results from a grid of insufficient spatial resolution.

JPEG JFIF (image compression)

Joint Photographic Experts Group. A collaborative specification of the CCITT and the ISO for image compression. The standard JPEG compression algorithm, which is used by ImageGear, is a lossy compression scheme - it loses data.

.JPG (file format extension)

Format originator: Joint Photographics Experts Group

kernel (image processing)

A small matrix of pixels, usually no bigger that 9x9, that is used as an operator during image convolution. The kernel is set prior to the convolution in a fashion that emphasizes a particular feature of the image. Kernels are often used as spatial filters, each one tuned to a specific spatial frequency that the convolution is intended to highlight.

See Also:

convolution (image processing).

Lempel Ziff Welch (data compression)

A dictionary-based image compression method with lossless performance that results in fair compression ratios. Most files are compressed at 2:1.

level (imaging)

See gain & level (imaging).

library (software)

A collection of software functions that can be called upon by a higher level program. Most libraries are collections of similar routines, for example, those used for graphical or image processing.

See Also:

DLL (Microsoft Windows)

Look-Up-Table (computer hardware)

A look-up-table or LUT is a continuous block of computer memory that computes the values of a function for one variable. The LUT is set up for the function's variable to be used as an address or offset into the memory block. The value that resides at this memory location becomes the function's output. Because the LUT values need only be initialized once, LUTs are very useful for image processing due to their inherent high speed.

LUT[pixel_value] = f(pixel_value)

LUTs come in various widths, usually in units of bits. An nxm bit LUT has 2n addresses or 256 stored values. Each value is 2m bits wide.

If the second dimension is left off it can be assumed to be equal to the first. In grayscale image processing, LUTs are commonly 8x8, and the bit widths are usually assumed.

A linear LUT, sometimes called a NOP LUT or pass through, is a LUT that is initialized to output the same values as the input. NOP_LUT[pixel_value] = pixel_value.

See palette (digital imaging).

lossless (image compression)

A method of image compression where there is no loss in quality when the image is uncompressed. The uncompressed image is mathematically identical to its original. Lossless compression is usually lower in compression ratio than lossy compression.

lossy (image compression)

A method of image compression where some image quality is sacrificed in exchange for higher compression ratios. The amount of quality degradation depends on the compression algorithm used and by a user-selected quality variable.

LUT (computer)

Look-Up-Table. See Look-Up-Table (computer hardware).

LUT transform (image processing)

A LUT transform is an image processing method that takes an image and passes each pixel, one at a time, through a pre-set LUT. Each new pixel is a function of one and only one pixel from the original image and is arranged in the same location.

Any image-processing algorithm that transforms a single pixel into another single pixel, both from the same location, can be performed quickly using a LUT.

Square_root_LUT[pixel_value] = sqrt(pixel_value)

See Also:

Look-Up-Table (computer hardware)

LZW (data compression)

Lempel Ziff Welch. See Lempel Ziff Welch (data compression).

machine vision

A sub-discipline of artificial intelligence that uses video cameras or scanners to obtain information about a given environment. Machine vision processes extract information from digital images about objects in the image. This is the opposite of computer graphics that takes various data describing objects in and produces an output image. Machine vision takes an image in and outputs some level of description about the objects in it, (i.e. color, size, brightness).

See Also:

image processing.

matrix operation (image processing)

See neighborhood process (image processing).

median filter (image processing)

An image spatial filtering operation based on an input pixel and its 8 neighbors. The resulting value is the median (5th from the sorted values). A median filter is often used to reduce spike or speckling noise from a grayscale image. It has the advantage over convolution smoothing - it better preserves edges.

morphing (image processing)

An imaging process where one image is gradually transformed into a second image, where both images previously existed. The result is a sequence of in-between images when played sequentially, as in a film loop show, give the appearance of the starting image being transformed to the second image.

Morphing is made up of a collection of image processing algorithms. The two major groups are: warps and blends. Do not confuse this with morphology.

MPEG (image compression)

Motion Pictures Experts Group. An ISO specification for the compression of digital-broadcast quality full-motion video and sound.

neighborhood process (image processing)

A class of image-processing routines that works on neighborhoods of pixels. Each pixel in the new image is computed as a function of the neighborhood of the pixel from the original pixel. The neighborhood ID is defined by a kernel that is set once for each image to be processed.

See Also:

point process (image processing)

(one) 1-bit image (digital imaging)

An image comprised of pixels that contain only a single bit of information. Each pixel is either on or off. Normally, "on" is white and "off" is black.

opening (image processing)

See MPEG (image compression).

overlay (imaging)

An image or sub-image that can be placed over a given image. The pixels from the original image are not altered but the overlay can be viewed as if they had been. Usually used to place temporary text and annotation marks, for example, arrows on a image.

packed bits (imaging)

A binary image is usually stored in computer memory (8 pixels per byte). In this case each byte is referred to as being filled with packed bits. This saves space but makes reading and writing any individual pixel harder since most computers cannot directly access memory in chunks smaller than a byte.

palette (digital imaging)

A digital image palette is a collection of 3 look-up-tables, or LUTs, that are used to define a given pixel's display color. One LUT is for red, one for green and one for blue. The number of entries in the LUTs depend on the width (in bits) of the image's pixels.

A palette image requires its palette in order to be displayed in a fashion that makes sense to the viewer. This is often the case for color 8-bit images. Without a palette describing what color each pixel needs for display, this type of image would most likely be displayed as randomly selected noise.

A grayscale palette is one where each of the 3 LUTs are linear. The output is whatever is input to them. Since each color component (R, G, B) is an equal value, any pixels input to them are displayed in a varying shade of gray.

See Also:

Look-Up-Table (computer hardware)

pattern recognition (imaging)

A sub-discipline of machine vision where images are searched for specific patterns. Optical character recognition or "OCR" is one type of pattern recognition, where images are searched for the letters of the alphabet.

.PCX (file format extension)

Format originator: ZSoft Corp.

450 Franklin Road Suite 100

Marietta, GA 30067

pixel (imaging)

An abbreviated version of the term PIcture (X) ELement. This is the most fundamental element of a digital image. A digital image is made up of rows and columns of points of light. Each indivisible point of light is a pixel. Each pixel in an image is addressed by its column (x) and its row (y) usually written as the coordinate pair (x, y). An 8-bit pixel can take on one of 256 values. A 24-bit pixel has 3, 8-bit components for each of the primary colors, red, green, and blue.

point process (image processing)

A class of image processing transforms where every pixel is taken, one at a time from an image, and mathematically transformed into a new value with no input from any other pixel in the image. A point process is a degenerative neighborhood process where the kernel is a matrix of pixels that is 1x1 or in other words a single pixel.

polar coordinates (imaging)

An alternative to the usual Cartesian method of addressing image pixels. Polar coordinates use the coordinate pair, angle and radius from an origin instead of column and row.

posterize (imaging)

A special effect that decreases the number of colors or grayscale colors in an image. The default image pixel contains 256 levels of gray or 256 levels of red, green, and blue. Using this effect reduces these numbers.

pseudocolor (image processing)

A method of assigning color to ranges of a grayscale image's pixel values. Most often used to highlight subtle contrast gradients or for visually quantifying pixel values. The applied color usually has no correspondence to the original scene. The colors are used only as a guide or highlight.

raster (imaging)

A term that describes a single row of a digital image. A raster image is made up of rows of pixels. This is opposed to vector images, where an image is made up of a list of polygon nodes. A raster is sometimes called a scan-line.

relative coordinates

Relative coordinates refer to position, as identified as the distance from a local origin.

render (imaging)

The process of displaying an image. The final and actual displayed image is said to be rendered.

resolution (imaging)

There are two types of resolution in digital images; spatial and intensity. Spatial resolution is the number of pixels per unit of length along the x and y axis. Intensity resolution is the number of quantized levels that a pixel can have.

RGB (imaging)

Red, Green, Blue. A triplet of numeric values that describe a color.

RGBQUAD

Red, Green, Blue, Quad. A set of four numbers used to describe a color. The forth number is always set to zero. This creates an efficient color LUT or palette. It is more efficient because most computers find multiplying by 4 easier then by 3, as is the case in an RGB triplet.

ROI (Accusoft image processing)

Region Of Interest. A region of interest or ROI is a specification and date structure that allows for the definition of arbitrarily shaped regions within a given image, often called sub-images. A ROI can be thought of as a place holder which remembers a location within an image. ROIs are of several types, each defined in a manor that makes sense for its type.

ROIs are either a rectangle (also called an AOI), square, circle, or a segment list. A rectangle is defined by any two points in the image. From these two points one and only one rectangle can be drawn. A square is defined by a single point and a single length. A circle is defined by its center and radius. A segment list is an arbitrary list of triplets (x, y, xlen); a single point and a length to the right.

Every point in an image is either inside or outside of a given ROI.

Most image processing functions in this package work only within a given ROI. The ROI can encompass the entire image.

See Also:

AOI (Image Processing)

scan line (imaging)

See raster (imaging)

screen coordinates (imaging)

Screen coordinates are those of the actual graphics display controller. The origin is almost always at the upper left-hand corner of the display.

See Also:

coordinates

segment (imaging)

A contiguous section of a raster line. It is defined in physical coordinates by the triplet of its left most point and length (x, y, length).

shear (image processing)

A skew is image distortion that often occurs when a scanner is sampling an image and the image slides to either side before the scan is complete. This has the effect of transforming squares into rhombuses.

special effects (image processing)

Any image processing transform that is applied mostly for its artistic value. Special effects include, wipes, transitions, barn doors, etc.

stretch intensity (image processing)

An image processing method that takes a given image and assures that the intensity distribution fills the entire range of possible values. An 8-bit image that is stretched always has at least one pixel with a value of zero and one of 255. The term comes from the before and after histogram of the given image. A stretch operation linearly stretches a histogram so that is ranges from the minimum pixel value to the maximum pixel value.

.TGA (file format extension)

Format originator: Truevision, Inc.

7340 Shadeland Station

Indianapolis, IN 46255

TIFF (file format)

Tagged Image File Format.

.TIF (file format extension)

Format originator: Aldus Corp

411 First Ave South

Seattle, WA 98104, and

Microsoft Corp

16011 NE 36th Way

Redmond, WA 98073

thumbnail (imaging)

A small copy of an image. Thumbnails are used to display many images on the screen at once.

transform (image processing)

An algorithm that takes an image, alters it, and outputs a new image. Sometimes written as `xform'.

See Also:

- geometric transform (image processing)
- neighborhood process (image processing)
- point process (image processing)

triplet (digital imaging)

Three numbers used together to represent a single quantity or location, for example, RGB or (x, y, z).

(twenty-four) 24 bit image (digital imaging)

A 24-bit image contains pixels made from RGB triplets.

video stream (video)

A sequence of still images that are transmitted and displayed in synchronous order that give the appearance of live motion.

warp (image processing)

A geometric image processing routine that distorts an image by spatially compressing and stretching regions.

.WMF (file format extension)

Format originator: Microsoft Corp

16011 NE 36th Way

Redmond, WA 98073

world coordinates

The real valued coordinates that make sense for the object, treating it as if it really exists. The world coordinates of a house on a map would be in miles or longitude and latitude. This is the opposite of screen, device or model coordinates.

.WPG (file format extension)

Format originator: WordPerfect Corp

(x, y)

A mathematical method for referring to a pixel from a digital image. Since most digital images are maintained as a Cartesian matrix of pixels, each pixel has a unique address that can be described as an x or horizontal displacement from the origin and a y or vertical displacement from the origin.

See Also:

coordinates

xform

Shorthand for transform.

YIQ (color imaging)

(Y) luminance, (I), (Q). YIQ is the color model used for U.S. commercial television. It was designed to be backwards compatible with the old black and white television sets. "Y" or luminance is a weighted average of the red, green, and blue that gives more weight to red and green than to blue. The I and Q contain the color components. Together they are called chromaticity.

(Z)

A mathematical method that refers to a pixel's intensity from a digital image. An image can be written as: I(x,y)=z

1.2.7.6.1 ImageGear Licensing and Deployment Kit Terminology

Access Key

The key provided to the end user for licensing the application. Uniquely identifies each license issued by Accusoft. You can only generate as many different access keys as the total number of deployment licenses purchased. One an access key needs to be associated with each end user's machine, you can choose to either: distribute access keys to end users explicitly, or use the Licensing Component to acquire access keys automatically behind the scenes.

Concurrent

Licenses that are co-used by a specific number of users and that are counted as 1 license.

Deployment Kit

A set of wizards and tools that help you license your applications.

Deployment Licensing Service

The Web service at Accusoft that handles all licensing requests.

Deployment Packaging Wizard

The tool included in the Deployment Kit that helps you package the appropriate ImageGear runtime components.

Deployment Pool

Each solution can have several deployment pools associated with it. A Pool is a set of licenses of the same type. When a pool is set up, the product features, the deployment model, and other information are associated with it. All the licenses from the same pool have the same attributes, except for hardware parameters (if bound to hardware parameters).

Deployment Proxy Service

The Web service running at your site for "proxying" between the Deployment Licensing Service and the Licensing Component running on the end user's system.

End User

The customer using your application.

End User Licensing Utility

The Web application you use to generate license keys based on hardware keys.

Hardware Key

The key dynamically constructed on the end user's machine by combining the access key and hardware parameters of the machine. Obtained on the end user's machine by combining the access key and the hardware parameters of the end user system. This typically happens during an installation process using the Licensing Component.

ImageGear Runtime Components

The ImageGear runtime components that are licensed for deployment with your application.

License Key

The key containing information about ImageGear licensed features and hardware parameters of the target machine (if bound to hardware parameters). Generated by the Deployment Licensing Service in exchange for the hardware key, and stored in the End User's system registry.

Licensing Component

A component that obtains an ImageGear license for the end user's machine.

Licensing Component Wrapper

A VBScript wrapper around the Licensing Component with methods for handling error result codes, etc. This wrapper serves as a template that can be modified as needed.

LPK

An LPK file is a file that licenses ImageGear ActiveX controls when running on a Microsoft Internet Explorer Web Browser. This file must be created for all ActiveX components that require licensing, regardless of manufacturer.

Named

Licenses that are counted as they are deployed to individual users.

Server License

Bound to hardware parameters. Therefore an individual node-locked License Key has to be generated for each system where the ImageGear-based solution is used. With this model, the Licensing Component running on the end user's machine can communicate directly to the Deployment Licensing Service but you also have the choice of setting up a Deployment Proxy Service at your site - in this case Licensing Component would communicate through this proxy service.

Server Licensing Utility

A standalone program that can be used to generate a license key given a configuration file.

Solution Key

A combination of 4 32-bit integers that are unique for each solution using Accusoft's technology. This key is generated and assigned to your solution by Accusoft.

Solution Name

The name of your ImageGear-based application.

User License

Not bound to hardware parameters. Therefore only one license key is required per ImageGear-based application.

Vendor

The Accusoft customer who is developing an ImageGear-based application (you).

Vendor Licensing Utility

The Web application you use to view deployment pools and obtain access keys.

1.2.7.7 Bibliography

Bibliography

Books

Brown, Wayne C., and Barry J. Shepherd. Graphics File Formats Reference and Guide. Greenwich, CT: Manning Publications, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. Windcrest Books, 1992.

Kay, David C. and John R. Levine. Graphics File Formats. 2d ed. Windcrest/McGraw-Hill, 1995.

Murray, James D., and William vanRyper. Encyclopedia of Graphics File Formats. Sebastapol, CA: O'Reilly & Associates, Inc., 1994.

Murray, James D. and William vanRyper. Encyclopedia of Graphics File Formats, 2d ed. Sebastopol, CA: O'Reilly & Associates, Inc., 1996.

Petzold, Charles. Programming Windows: The Microsoft Guide to Writing Applications for Windows 3. Redmond, WA: Microsoft Press, 1990.

Articles

Wegner, Tim. "Coding for PNG Graphics", PC Techniques (Feb/Mar 1996): 32-38.

Other

"Graphic Image Format FAQ 3-4." James D. Murray, 1994-1996.

PBM, PGM, PPM, PNM Specifications by Jef Poskanzer, copyright " 1989, 1991.

PNG (Portable Network Graphics), tenth draft. Page 5, copyright Thomas Boutell, May 1995.

1.3 API Reference Guide

The ImageGear Professional API Reference Guide provides detailed information about each function or control parameter of the ImageGear Components. This information includes each function's calling sequence, arguments, use, possible return values, and supported raster image formats as well as other useful information.

1.3.1 Core Component API Reference

This section provides detailed information about the ImageGear Core component API in the following sections:

- Core Component Data Types Reference
- Core Component Functions Reference
- Core Component Callback Functions Reference
- Core Component Structures Reference
- Core Component Enumerations Reference

1.3.1.1 Core Component Data Types Reference

The following are data types that may appear in an ImageGear IG_ ...() function call.

- AT_BOOL
- AT_CHAR
- AT_DIMENSION
- AT_ERRCOUNT
- AT_LMODE
- AT_MODE
- AT_PIXEL
- AT_PIXPOS
- AT_WCHAR
- HIGEAR

1.3.1.1.1 AT_BOOL

An integer that is interpreted as FALSE if 0 and TRUE if non-0.

1.3.1.1.2 AT_CHAR

Unsigned 8-bit integer. It is often used to represent an ANSI character.

1.3.1.1.3 AT_DIMENSION

Type usually used for the width in pixels or height in rows of an image in memory or of a display area on a display device.

1.3.1.1.4 AT_ERRCOUNT

Type of value returned by most ImageGear functions. It is an integer equal to the number of errors placed on the ImageGear error stack during execution of the function and any lower level functions called.

1.3.1.1.5 AT_LMODE

Type of a 32-bit constant (such as IG_SAVE_BMP_RLE), or of a variable containing such a value, used to declare what mode of operation a function should use. Most variables have constants defined for them in accucnst.h. Occasionally, a variable of this type will be a Windows constant.

1.3.1.1.6 AT_MODE

Type normally used for a constant that is used to specify an option in a call to an ImageGear function. Examples are IG_CONTRAST_PIXEL or IG_ASPECT_DEFAULT. This type is used when the constant's value can never require more than 16 bits.

1.3.1.1.7 AT_PIXEL

A BYTE, usually a byte in an image bitmap.

1.3.1.1.8 AT_PIXPOS

A 64-bit integer value specifying a pixel x or y coordinate.

1.3.1.1.9 AT_WCHAR

Unsigned 16-bit integer. It is used to represent a wide character (UTF-16 encoded).

1.3.1.1.10 HIGEAR

Handle of a comprehensive ImageGear data structure that defines an image along with its display attributes, Look-Up Tables, DIB, and other data necessary for maintaining the image and providing access to its pertinent data for your application program.

1.3.1.2 Core Component Functions Reference

This section describes each function supported by ImageGear Core component, arranged in alphabetical order within functional groups.

1.3.1.2.1 ASCII Functions

This section provides information about the ASCII group of functions.

- IG_ascii_import
- IG_ascii_page_width_get

1.3.1.2.1.1 IG_ascii_import

This function loads an ASCII (.TXT) file into ImageGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_ascii_import (
        const LPSTR lpszFileName,
        UINT nPageNumber,
        LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Set to the path/filename of the text file to load.
nPageNumber	UINT	Set to the number of the page to load; if not a multi-page file, set to 1.
lphIGear	LPHIGEAR	A far pointer that returns a HIGEAR handle for your newly loaded image.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Specify the path/filename of the file and which page of the file (if it is a multi-page file) you would like to load. ImageGear then returns a HIGEAR handle to the newly loaded image.

To set the control parameters of the loaded file, use IG_fltr_ctrl_set(). The TXT filter control parameters are listed in the section TXT (ASCII Text).

You may also use IG_load_file() to load an ASCII file.

1.3.1.2.1.2 IG_ascii_page_width_get

This function returns the width of an ASCII file that has not been loaded yet.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_ascii_page_width_get (
        const LPSTR lpszFileName,
        LPUINT lpPageWidth
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Set to the name of the ASCII file from which to get the width.
lpPageWidth	LPUINT	A far pointer that returns a value of type UINT indicating the width of the file in thousands of an inch.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR hIGear;         /* HIGEAR handle of image   */
AT_ERRCOUNT nErrcount;      /* total # of ImageGear errors on the stack*/
UINT pageWidth;     /* returns the width of the ASCII file */
UINTpointSize; /* to get and possibly set the point size of the font */
AT_REAL reduction;    /* percentage of reduction possibly needed on the point size before
the image is loaded with a width of 8.5 inches   */
nErrcount = IG_ascii_page_width_get("Hamlet.txt", &pageWidth);
if (pageWidth > 8500)
/* if the width of the unloaded page is greater than 8.5 inches, we will resize it to 8.5
inches before loading it. However, to avoid cropping any words, we will first reduce the
point size by the percentage needed to make each line fit on an 8.5 inch page */
{
        reduction = 8500/pageWidth;
        pointSize = (UINT)(pointSize * reduction + 0.5);
        nErrcount =
                IG_fltr_ctrl_set(IG_FORMAT_TXT, "POINT_SIZE",
                (LPVOID)pointSize, sizeof(pointSize));
        nErrcount =
                IG_fltr_ctrl_set(IG_FORMAT_TXT, "PAGE_WIDTH",
                (LPVOID)8500, sizeof(nPageWidth));
}
nErrcount = IG_ascii_import("Hamlet.txt", 1, &hIGear);
```

Remarks:

Specify the path/filename of an ASCII file, and this function will return the width in thousands of an inch. For example, if the file has a width of 8.5 inches, this function returns the value 8500.

The width of the page (as well as many other attributes) can be set before the page is loaded. See the description for IG_ascii_import()for the full list of attributes that can be determined prior to loading.

As an alternative to this function, you can use IG_fltr_ctrl_get() with the control parameter argument "PAGE_WIDTH".

1.3.1.2.2 Callback Register Functions

This section provides information about the Callback Register group of functions.

- IG_batch_CB_register
- IG_file_IO_register
- IG_mem_CB_register
- IG_status_bar_CB_register

1.3.1.2.2.1 IG_batch_CB_register

This function registers one of two available batch callback functions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_batch_CB_register(
        LPVOID lpfnBatchCB,
        AT_MODE nCBType,
        LPVOID lpPrivate
);
```

Arguments:

Name	Type	Description
lpfnBatchCB	LPVOID	A far pointer to the scan callback function you would like to register.
nCBType	AT_MODE	Set to the type of callback being registered. Use one of the following ImageGear-defined constants: ● IG_BATCHCB_BEFORE_OPEN ● IG_BATCHCB_BEFORE_SAVE
lpPrivate	LPVOID	Optional pointer to from which to pass and receive data. Set this to NULL if this is not required.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

See example the code for the LPFNIG_BATCH_BEFORE_SAVE and LPFNIG_BATCH_BEFORE_OPEN functions.

Remarks:

This function registers one of two available batch callback functions.

- LPFNIG_BATCH_BEFORE_OPEN is called before a file is opened, allowing you to get the file name and correct some settings. For example, some multimedia formats and PDF files require you to get the file name before converting a page.
- LPFNIG_BATCH_BEFORE_SAVE is called before an image file is saved, allowing you to correct an image before saving it. For example, you might want to rotate an image before saving it.

1.3.1.2.2.2 IG_file_IO_register

This function registers your own functions to be called to do Reads, Writes, and Seeks during image file transfers.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_file_IO_register (
        LPFNIG_READ lpfnReadFunc,
        LPFNIG_WRITE lpfnWriteFunc,
        LPFNIG_SEEK lpfnSeekFunc
);
```

Arguments:

Name	Type	Description
lpfnReadFunc	LPFNIG_READ	Far pointer to your function to be called for READs.
lpfnWriteFunc	LPFNIG_WRITE	Far pointer to your function to be called for WRITEs.
lpfnSeekFunc	LPFNIG_SEEK	Far pointer to your function to be called for SEEKs.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
LPFNIG_READ              MyReadFunc;      /* To be called for file READs */
{
HIGEAR          hIGear;     /* Will hold HIGEAR handle of image  */
 ...
IG_file_IO_register ( MyReadFunc, NULL, NULL );/* Register it */
 ...
IG_load_file ( "picture.bmp", &hIGear );
 ...
}
/* This will be called for each read during the above Load:   */
LONG  ACCUAPI MyReadFunc ( LONG fd, LPBYTE lpBuffer,
        LONG lNumToRead )
{
LONG     nNumActuallyRead;
 ...   /* May transfer bytes to buffer in any way     */
return  nNumActuallyRead; /* Return count, or -1 for error  */
}
```

Remarks:

An argument should be NULL if you want ImageGear to perform that operation. See also the descriptions for typedefs LPFNIG_READ, LPFNIG_WRITE, and LPFNIG_SEEK.

1.3.1.2.2.3 IG_mem_CB_register

This function registers your own callback functions to be called to do large memory allocations, memory reallocations, and memory freeing.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mem_CB_register(
        LPFNIG_MEM_ALLOC lpfnAllocFunc,
        LPFNIG_MEM_REALLOC lpfnReAllocFunc,
        LPFNIG_MEM_FREE lpfnFreeFunc
);
```

Arguments:

Name	Type	Description
lpfnAllocFunc	LPFNIG_MEM_ALLOC	Far pointer to your function to be called for memory allocations.
lpfnReAllocFunc	LPFNIG_MEM_REALLOC	Far pointer to your function to be called for memory reallocations.
lpfnFreeFunc	LPFNIG_MEM_FREE	Far pointer to your function to be called to free memory free.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Memory Alloc callback function definition                          */
LPBYTE  ACCUAPI MyMemAlloc(DWORD dwSize)/* number of bytes to alloc    */
{
        /* Put your own memory allocation code here */
        return( buffer);
};
/* Memory ReAlloc callback function definition                        */
LPBYTE ACCUAPI MyMemReAlloc( LPBYTE lpBuffer, DWORD
        dwSize)
{
        /* Put your own memory reallocation code here */
        return( lpBuffer);
};
/* Memory Free callback function definition   */
LPBYTE  ACCUAPI MyMemFree(LPBYTE lpBuffer)
{
        /*Put your free-the-memory code here */
        return NULL;
};
/* Registration Example  */
/* Example one  */
/* Register your own callback functions for all memory routines  */
nErrcount = IG_mem_CB_register(MyMemAlloc, MyMemReAlloc, MyMemFree);
/* Example two */
/* Supply callbacks for memory alloc only  */
nErrcount = IG_mem_CB_register(MyMemAlloc, NULL, NULL);
```

Remarks:

As shown in the prototype, your memory callback functions must be of types LPFNIG_MEM_ALLOC, LPFNIG_MEM_REALLOC and LPFNIG_MEM_FREE.

Your memory allocation functions will only be used when large allocations (allocations greater than 1024) are performed.

Set any of the three arguments to NULL if you want ImageGear to use its own definition for these functions.

1.3.1.2.2.4 IG_status_bar_CB_register

This function establishes a status bar callback function to be called by ImageGear during load, save, and print operations.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_status_bar_CB_register (
        LPFNIG_STATUS_BAR lpfnStatusBar,
        LPVOID lpPrivate
);
```

Arguments:

Name	Type	Description
lpfnStatusBar	LPFNIG_STATUS_BAR	Far pointer to a function to be established as your status bar callback function. ImageGear will call this function once for each raster (pixel row) processed during load, save, print, and image processing operations. The argument list and return value of this function must be as shown in the definition of function type LPFNIG_STATUS_BAR.
lpPrivate	LPVOID	Far pointer to private data of your own choosing.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
LPFNIG_STATUS_BAR    MyStatusBarCallback;    /* Declare type of function */
{
static DWORD      dwPrivateFlags;
 ...
IG_status_bar_CB_register ( MyStatusBarCallback, (LPVOID) &dwPrivateFlags );
```

Remarks:

ImageGear will call the named function once for each raster (row) processed, transmitting the Y position of that raster, the total number of rasters involved in the transfer, and the value of lpPrivate (pointer to your private data area). Your callback function can use this data to display a status bar showing percent completion, or for any other purpose.

To change to a different status bar function, or to change to a different private data area, call IG_status_bar_CB_register() again with your new callback function name and/or private data area address.

To disable status bar callbacks, call IG_status_bar_CB_register() with argument lpfnStatusBar = NULL.

See also the description for function type LPFNIG_STATUS_BAR.

1.3.1.2.3 Clipboard Functions

This section provides information about the Clipboard group of functions.

- IG_clipboard_copy
- IG_clipboard_cut
- IG_clipboard_dimensions
- IG_clipboard_paste
- IG_clipboard_paste_available
- IG_clipboard_paste_available_ex
- IG_clipboard_paste_merge
- IG_clipboard_paste_merge_ex
- IG_clipboard_paste_op_get
- IG_clipboard_paste_op_set

1.3.1.2.3.1 IG_clipboard_copy

This function copies the specified portion of the image to the system clipboard.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_copy (
        HIGEAR hIGear,
        const LPAT_RECT lprcRectToCopy
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the current image.
rcRectToCopy	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image that is to be copied to the clipboard.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

The copied pixels will be converted to 1-, 4-, 8-bit indexed or 24-bit RGB format for copying to the clipboard.

Example:

```
HIGEAR       hIGear;        /* HIGEAR handle of image  */
AT_RECT      rcRectToCopy;  /* Rectangle of image to copy to clipboard */
AT_DIMENSION nImageWidth;    /* Width of image  */
AT_DIMENSION nImageHeight;  /* Height of image  */
/* Copy bottom half of image to system clipboard:  */
rcRectToCopy.left   = 0;
rcRectToCopy.top    = nImageHeight / 2;
rcRectToCopy.right  = nImageWidth - 1;
rcRectToCopy.bottom = nImageHeight - 1;
IG_clipboard_copy ( hIGear, &rcRectToCopy );
```

Remarks:

If rcRectToCopy = NULL, the entire image will be copied.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.3.2 IG_clipboard_cut

This function "cuts away" a portion of an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_cut (
        HIGEAR hIGear,
        const LPAT_RECT lprcRegion,
        const LPAT_PIXEL lpPixel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of current image.
lprcRegion	const LPAT_RECT	A long pointer to a rectangular region of the image, which should be copied to the clipboard and removed from the image.
lpPixel	const LPAT_PIXEL	A long pointer to a pixel value you would like to be used to fill in the area out of which the region has been cut.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

The copied pixels will be converted to 1-, 4-, 8-bit indexed or 24-bit RGB format for copying to the clipboard.

Example:

```
HIGEAR         hIGear;     /* HIGEAR handle of image  */
AT_ERRCOUNT    nErrcount;  /* # of IG errors on the stack  */
AT_RECT        rcRegion;   /* Rectangular region to cut from image */
AT_PIXEL       pixel       /* Pixel value used when filling cut regions */
pixel = 0;
/* For a currently loaded 1-bit image where black = 0, set the cut area to black*/
nErrcount = IG_clipboard_cut ( hIGear, &rcRegion, &pixel );
```

Remarks:

The cut portion is copied to the clipboard with its original pixel values, while in the displayed image, that rectangle is replaced by a pixel value as specified by lpPixel. The color used is usually black or white. You can restore the image to its original composition by calling IG_clipboard_paste_merge_ex(). You then set the x and y positions of the upper left-hand corner arguments to PIXPOS left and PIXPOS right of the image rectangle defined by lprcRegion. If you save the image after a call to this function, it will be saved with the cut.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE (i.e. an NRA mask is active), and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROIdefined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask()

functions for more details.

For 24-bit images, lpPixel must point to 3 bytes where the first byte is red, the second - green and the third - blue. For all other bit depths, lpPixel must point to a single byte.

1.3.1.2.3.3 IG_clipboard_dimensions

This function obtains the dimensions of the image currently in the system clipboard.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_dimensions (
        LPAT_DIMENSION lpWidth,
        LPAT_DIMENSION lpHeight,
        LPUINT lpBitsPerPixel
);
```

Arguments:

Name	Type	Description
lpWidth	LPAT_DIMENSION	Far pointer to a variable of type AT_DIMENSION to receive the width in pixels of the image currently in the system clipboard.
lpHeight	LPAT_DIMENSION	Far pointer to a variable of type AT_DIMENSION to receive the height in rows of the image currently on the system clipboard.
lpBitsPerPixel	LPUINT	Far pointer to a variable of type UINT to receive the bit depth, in bits per pixel, of the image currently on the system clipboard.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_DIMENSION   nWidth, nHeight;   /* holds the images's width and height   */
UINT           nBpp;              /* holds the bits per pixel   */
AT_ERRCOUNT    nErrcount; /* holds athe returned error count */
BOOL           bPasteAvail;     /* TRUE if a pasteable image is on the clipboard */
 /* If a pasteable image is on the clipboard, get its dimensions: */
IG_clipboard_paste_available ( &bPasteAvail);
if ( bPasteAvail )
  {
nErrcount = IG_clipboard_dimensions ( &nWidth, &nHeight, &nBpp );
  }
```

Remarks:

Prior to calling this function you should call IG_clipboard_paste_available_ex(), to verify that there is an image that can be pasted in the system clipboard.

1.3.1.2.3.4 IG_clipboard_paste

This function creates a HIGEAR image by pasting the image on the system clipboard.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste (
        LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lphIGear	LPHIGEAR	Far pointer to a variable of type HIGEAR, to receive the HIGEAR handle of the image created by this operation.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB – 1, 4, 8 bpp;
RGB – 24 bpp.

Example:

```
HIGEAR hIGear; /* Receives the HIGEAR handle the image created  */
BOOL bPasteAvail;  /* TRUE if a pasteable image is present  */
AT_ERRCOUNT   nErrcount;    /* Holds the returned error count*/
IG_clipboard_paste_available ( &bPasteAvail);
if ( bPasteAvail )
/* Create HIGEAR image from contents of system clipboard:*/
        { nErrcount = IG_clipboard_paste ( &hIGear );
        if ( nErrcount )  { ... } /* Process any errors ...*/ }
```

Remarks:

Prior to calling this function, call IG_clipboard_paste_available_ex() to verify that there is a paste-able image in the clipboard.

1.3.1.2.3.5 IG_clipboard_paste_available

This function retrieves whether there is compatible data available in the clipboard.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_available(
      LPAT_BOOL lpPasteStatus
);
```

Arguments:

Name	Type	Description
lpPasteStatus	LPAT_BOOL	Pointer indicating where to return the boolean value that indicates whether clipboard data is available.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The compatible data types supported on Windows systems are: CF_DIB, CF_BITMAP, CF_ENHMETAFILE, and CF_METAFILEPICT.

To determine the number of errors currently on the error stack use IG_error_check. After fetching all error information you need using IG_error_get, use IG_error_clear to clear the stack.

See Also:

IG_clipboard_paste_available_ex

1.3.1.2.3.6 IG_clipboard_paste_available_ex

This function tells you whether the clipboard contains a valid image or region of interest (ROI) that can be pasted into an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_available_ex (
        LPBOOL lpPasteStatus,
        LPAT_MODE lpRegionType
);
```

Arguments:

Name	Type	Description
lpPasteStatus	LPBOOL	A far pointer that returns TRUE if the clipboard contains a region of interest that can be pasted into an image.
lpRegionType	LPAT_MODE	A far pointer that returns the type of region stored in the clipboard. Currently the valid values are IG_REGION_IS_RECT, IG_REGION_IS_NON_RECT, IG_REGION_IS_NOT_AVAIL.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR hIGear;
BOOL bNRpasteAvail; /* TRUE if a pasteable image is on clipboard */
AT_MODE nRegionType; /* type of region on the clipboard */
nErrcount = IG_clipboard_paste_available_ex(&bNRpasteAvail, &nRegionType);
if (bNRpasteAvail)
        (...)
```

Remarks:

It also returns the type of region contained in the clipboard: rectangular, non-rectangular, or not available. If the clipboard does contain a valid ROI, lpPasteStatus returns TRUE; if lpPasteStatus returns FALSE, the region type returned is IG_REGION_IS_NOT_AVAIL.

A return value of FALSE does not necessarily mean that the clipboard is empty. It could mean that the clipboard contains non-valued ROI data or text, or that it contains multimedia data.

To paste a rectangular or non-rectangular ROI into the current image, call IG_clipboard_paste_merge_ex().

1.3.1.2.3.7 IG_clipboard_paste_merge

This function retrieves compatible media from the clipboard, if available, and "places" or merges the clipboard media into the specified image at the specified coordinates.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_merge(
   HIGEAR hIGear,
   AT_PIXPOS nLeftPos,
   AT_PIXPOS nTopPos
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image into which the clipboard media will be merged.
nLeftPos	AT_PIXPOS	X-coordinate of the hIGear to which the clipboard media will be merged.
nTopPos	AT_PIXPOS	Y-coordinate of the hIGear to which the clipboard media will be merged.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The compatible data types supported on Windows systems are: CF_DIB, CF_BITMAP, CF_ENHMETAFILE, and CF_METAFILEPICT. Compatible data must be available on the clipboard for this API to be successful.

To determine the number of errors currently on the error stack use IG_error_check. After fetching all error information you need using IG_error_get, use IG_error_clear to clear the stack.

See Also:

IG_clipboard_paste_merge_ex

1.3.1.2.3.8 IG_clipboard_paste_merge_ex

This function pastes a rectangular or non-rectangular clipboard image into the HIGEAR image that you specify.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_merge_ex(
        HIGEAR hIGear,
        AT_PIXPOS nLeftPos,
        AT_PIXPOS nTopPos
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Set to the HIGEAR handle of the image in which to merge the clipboard contents.
nLeftPos	AT_PIXPOS	X position in HIGEAR image at which to place the upper left corner of the clipboard image when merging.
nTopPos	AT_PIXPOS	Y position in HIGEAR image at which to place the upper left corner of the clipboard image when merging.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB – 1, 4, 8 bpp;
RGB – 24 bpp.

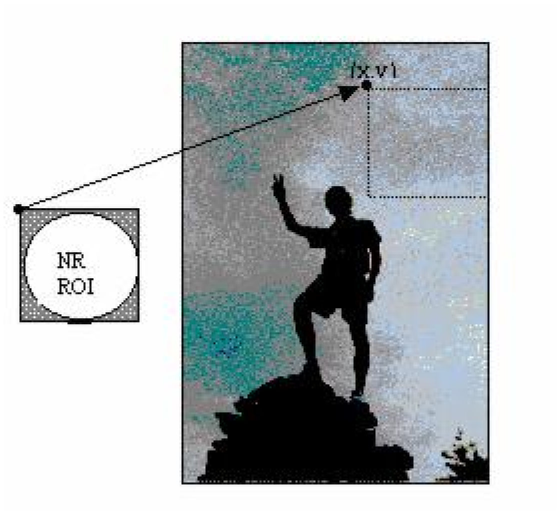
Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR  hIGear;
BOOL    bNRpasteAvail;
AT_PIXPOS       xpos, ypos;
AT_MODE         nRegionType;
nErrcount = IG_clipboard_paste_available_ex(&bNRpasteAvail, &nRegionType);
nErrcount = IG_clipboard_paste_merge_ex(hIGear, xpos, ypos);
```

Remarks:

To check if there is a paste-able image on the clipboard, call IG_clipboard_paste_available_ex().

nLeftPos and nTopPos set the (x,y) coordinates of the upper-left corner of the bounding rectangle of the original image. See image below. The image has an (x,y) location within it at which the upper left-corner of the bounding rectangle will be placed. The white circle enclosed within a gray rectangle represents the non-rectangular ROI image stored in the clipboard, where the shaded area represents the transparent area between the bounding rectangle and the non-rectangular ROI. When the clipboard image is merged, only the circle will appear on the image.



Use IG_clipboard_paste_op_set() to tell ImageGear what kind of merge operation to perform.

This function will automatically call IG_clipboard_paste_available_ex() to confirm that there is data available. Therefore, it is not mandatory to call IG_clipboard_paste_available_ex() before making this call, unless you are interested in knowing the type of region contained in the clipboard.

1.3.1.2.3.9 IG_clipboard_paste_op_get

This function returns the current paste-merge operation that will be used when an image from the clipboard is merged into the currently loaded image using IG_clipboard_paste_merge_ex().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_op_get (
        HIGEAR hIGear,
        LPAT_MODE lpOperation
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpOperation	LPAT_MODE	A long pointer to an integer constant of type AT_MODE. This will return the current setting for the kind of paste merge operation that will be performed.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image    */
AT_MODE nOperation;  /* current setting for paste-merge operation */
AT_ERRCOUNT nErrcount    /* # of IG errors currently on the stack  */
nErrcount = IG_clipboard_paste_op_get (hIGear, &nOperation);
```

Remarks:

See the description of IG_clipboard_paste_op_set() for the list of possible settings.

1.3.1.2.3.10 IG_clipboard_paste_op_set

This function sets the kind of operation to use for future calls to IG_clipboard_paste_merge_ex().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_clipboard_paste_op_set (
        HIGEAR hIGear,
        AT_MODE nOperation
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image that is in the system.
nOperation	AT_MODE	An integer constant of type AT_MODE, which will be used in future calls to IG_clipboard_paste_merge_ex().

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR       hIGear;         /* HIGEAR handle of image      */
AT_ERRCOUNT  nErrcount       /* # of IG errors on the stack */
nErrcount = IG_clipboard_paste_op_set( hIGear, IG_ARITH_AND );
```

Remarks:

nOperation is an integer constant of type AT_MODE that is defined in accucnst.h. Here are the possible settings and what kind of operation each one will perform on the values of the merging pixels:

IG_ARITH_ADD	Img1 = Img1 + Img2
IG_ARITH_SUB	Img1 = Img1 - Img2
IG_ARITH_MULTI	Img1 = Img1 * Img2
IG_ARITH_DIVIDE	Img1 = Img1 / Img2
IG_ARITH_AND	Img1 = Img1 & Img2
IG_ARITH_OR	Img1 = Img1 | Img2
IG_ARITH_XOR	Img1 = Img1 ^ Img2
IG_ARITH_ADD_SIGN_CENTERED	Img1 = Img1 + SC_Img2
IG_ARITH_NOT	Img1 = ~Img1
IG_ARITH_OVER	Img1 = Img2

You can also set nOperation to 0, which is the default. This will cause the image in the clipboard to just be copied over the currently loaded image - no merging of intersecting pixel values will occur.

1.3.1.2.4 Color Space Options Functions

This section provides information about the Color Space Options group of functions.

- IG_color_space_level_get
- IG_color_space_level_set

1.3.1.2.4.1 IG_color_space_level_get

This function has been deprecated and will be removed from the public API in a future release. Please use IG_image_colorspace_convert instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_color_space_level_get(
        AT_MODE nColorSpaceID,
        LPAT_MODE lpnSupportLevel
);
```

Arguments:

Name	Type	Description
nColorSpaceID	AT_MODE	Set this to an AT_MODE constant for the type of color space of which you would like to get the support level setting. The names of the appropriate constants begin with IG_COLOR_SPACE_ prefix.
lpnSupportLevel	LPAT_MODE	A far pointer that returns the current setting level for the color space specified in the first argument. See IG_color_space_level_set().

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT                      nErrcount;
HIGEAR                  hIGear;
AT_MODE                 nSupportLevel;
nErrcount = IG_color_space_level_get(IG_COLOR_SPACE_CMYK, &nSupporLevel);
```

Remarks:

This function queries the current option level setting for the color space that you specify.

You must supply nColorSpaceID with a constant of type AT_MODE from accucnst.h that specifies the color space you wish to query. The second argument will return an ImageGear constant that tells you the current option level setting for the color space.

ImageGear fully supports the loading and saving of TIFF-CMYK images. A CMYK image will only be converted to RGB for the purpose of display. CMYK is a color scheme designed for printing and cannot be used for screen display.

For more details see the description for IG_color_space_level_set().

1.3.1.2.4.2 IG_color_space_level_set

This function has been deprecated and will be removed from the public API in a future release. Please use IG_image_colorspace_convert instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_color_space_level_set(
        AT_MODE nColorSpaceID,
        AT_MODE nSupportLevel
);
```

Arguments:

Name	Type	Description
nColorSpaceID	AT_MODE	Set this to an AT_MODE constant for the type of color space you would like to set the support level setting for. For example: IG_COLOR_SPACE_CMYK.
nSupportLevel	AT_MODE	Set this to an AT_MODE constant for level of support for the color space specified in the first argument. For CMYK support, the possible settings are IG_CONVERT_TO_RGB or IG_FULL_SUPPORT.

ImageGear fully supports the loading and saving of CMYK images. A CMYK image will only be converted to RGB for the purpose of display. CMYK is a color scheme designed for printing and cannot be used for screen display.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT      nErrcount;
HIGEAR  hIGear;
nErrcount= IG_color_space_level_set(IG_COLOR_SPACE_CMYK, IG_COLOR_FULL_SUPPORT);
```

Remarks:

This function allows you to set the option level setting for the color space that you specify.

You must supply nColorSpaceID with a constant of type AT_MODE from accucnst.h that specifies the color space you wish to query, and nSupportLevel with a constant of type AT_MODE that specifies the level of support you would like your application to provide.

The CMYK color space is supported using the following settings:

IG_CONVERT_TO_RGB	Loads a CMYK image and converts it to RGB.
IG_FULL_SUPPORT	Full support for loading and saving CMYK images.

1.3.1.2.5 Component Manager Functions

This section provides information about the Component Manager group of functions.

- IG_comm_comp_attach
- IG_comm_comp_check
- IG_comm_comp_list
- IG_comm_entry_request
- IG_comm_function_call

1.3.1.2.5.1 IG_comm_comp_attach

This function allows you to attach ImageGear component defined by lpCompName to the main ImageGear module.

Declaration:

```
AT_ERRCODE  LACCUAPI IG_comm_comp_attach (
        LPCHAR lpCompName
);
```

Arguments:

Name	Type	Description
lpCompName	LPCHAR	The Name of ImageGear Component to be linked with main ImageGear module.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
...
#include "i_ART.h"
...
/*  Initialize ART component   */
IG_comm_comp_attach( "ART" );
...
```

Remarks:

By default, ImageGear searches for components in the same directory where main ImageGear module is located. You can specify a different path to the folder containing component modules using global parameters API function IG_gctrl_item_set() and "COMM.PATH" global parameter.

See Also:

ImageGear Components

Global Control Parameters

1.3.1.2.5.2 IG_comm_comp_check

This function allows you to check if the ImageGear component defined by lpCompName argument is currently attached or not.

Declaration:

```
AT_BOOL ACCUAPI IG_comm_comp_check(
        LPCHAR lpCompName
);
```

Arguments:

Name	Type	Description
lpCompName	LPCHAR	The Name of ImageGear Component attached to the main ImageGear module.

Return Value:

TRUE - if component is attached successfully. FALSE - if not.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
bFoundLZW = IG_comm_comp_check("LZW");
if( bFoundLZW )
{
EnableMenuItem( GetMenu( hWnd ), ID_FILE_SAVE_INTERLIVED, MF_ENABLED|MF_BYCOMMAND );
EnableMenuItem( GetMenu( hWnd ), ID_FILE_SAVE_NONINTERLIVED, MF_ENABLED|MF_BYCOMMAND );
IG_fltr_ctrl_get(IG_FORMAT_GIF, "INTERLACE", FALSE, NULL, NULL, (LPVOID)&bInterlaced,
sizeof(&bInterlaced));
CheckMenuItem(GetMenu(hWnd), ID_FILE_SAVE_INTERLIVED, MF_BYCOMMAND |
bInterlaced?MF_CHECKED:MF_UNCHECKED);
CheckMenuItem(GetMenu(hWnd), ID_FILE_SAVE_NONINTERLIVED, MF_BYCOMMAND |
bInterlaced?MF_UNCHECKED:MF_CHECKED);
}else
{
EnableMenuItem( GetMenu( hWnd ), ID_FILE_SAVE_INTERLIVED, MF_GRAYED|MF_BYCOMMAND );
EnableMenuItem( GetMenu( hWnd ), ID_FILE_SAVE_NONINTERLIVED, MF_GRAYED|MF_BYCOMMAND );
}
```

Remarks:

If component is attached it returns TRUE, if not - FALSE.

See also the section ImageGear Components.

1.3.1.2.5.3 IG_comm_comp_list

This function allows you to obtain information about currently loaded components.

Declaration:

```
AT_ERRCODE  ACCUAPI  IG_comm_comp_list(
        LPUINT* lpnCount,
        UINT nIndex,
        LPCHAR lpComp,
        DWORD dwCompSize,
        LPUINT lpnRevMajor,
        LPUINT lpnRevMinor,
        LPUINT lpnRevUpdate,
        LPCHAR lpBuildDate,
        UINT nBDSize,
        LPCHAR lpInfoStr,
        UINT nISSize
);
```

Arguments:

Name	Type	Description
lpnCount	LPUINT*	OUT: The number of attached components.
nIndex	UINT	IN: The index of component from the list.
lpComp	LPCHAR	OUT: The buffer where to return the name of component specified by nIndex index.
dwCompSize	DWORD	IN: The size of lpBuffer in bytes.
lpnRevMajor	LPUINT	Far pointer to an INT variable in which will be stored the Major version number of the version of the Component specified by nIndex.
lpnRevMinor	LPUINT	Far pointer to an INT variable in which will be stored the Minor version number of the version of the Component specified by nIndex.
lpnRevUpdate	LPUINT	Far pointer to an INT variable in which will be stored the Update (bug fix) number, reflecting any updates you have received and installed in this version of the Component specified by nIndex.
lpBuildDate	LPCHAR	The buffer where to return the build date of the current version of the Component specified by nIndex. The return value is a string in the format "Mmm dd yyyy", such as "Jul 04 2010."
nBDSize	UINT	The size of buffer where lpBuildDate is returned.
lpInfoStr	LPCHAR	The buffer where to return the info string about the Component specified by nIndex.
nISSize	UINT	The size of buffer where lpInfoStr is returned.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

First argument returns actual number of attached components. nIndex specifies the index of the component in the components list which name is copied into lpBuffer. The rest of parameters return information about component specified by nIndex.

1.3.1.2.5.4 IG_comm_entry_request

This function is used to get pointer to the function from component with given name.

Declaration:

```
AT_ERRCODE  ACCUAPI  IG_comm_entry_request(
        LPCHAR lpEntryName,
        LPAFT_ANY* lpFuncPtr,
        LPCHAR lpReason
);
```

Arguments:

Name	Type	Description
lpEntryName	LPCHAR	The full name of entry "<Comp_name>.<Func_name>", where <Comp_name> is a component name where function is exported, and <Func_name> is a name of function.
lpFuncPtr	LPAFT_ANY*	Pointer where to return pointer to requested function.
lpReason	LPCHAR	Optional parameter to specify string description of the reason for this request.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Usually this function is used with component public header i_<COMP_NAME>.h, where actual type of function is declared.

See also the section ImageGear Components.

1.3.1.2.5.5 IG_comm_function_call

This function is used to call function from a component.

Declaration:

```
LONG  CACCUAPI   IG_comm_function_call(
        LPCHAR lpEntryName,
        ...
);
```

Arguments:

Name	Type	Description
lpEntryName	LPCHAR	The name of entry in GFT to call.
...		Additional parameters.

Supported Raster Image Formats:

This function does not process image pixels.

Return Value:

Returns a LONG indicating the requested component function.

Example:

See the example in Component Manager API section of the Using ImageGear chapter.

Remarks:

Usually this function is not used directly, but it is used in macro declarations defined in component public headers i_<COMP_NAME>.h.

See also the section ImageGear Components.

1.3.1.2.6 Color Profile Management Functions

This section provides information about the Color Profile Management group of functions.

- IG_cpm_image_embedded_profile_check
- IG_cpm_image_profile_get
- IG_cpm_image_profile_set
- IG_cpm_profile_get
- IG_cpm_profile_set
- IG_cpm_profiles_reset

1.3.1.2.6.1 IG_cpm_image_embedded_profile_check

This function checks to see whether the image has embedded color profile.

Declaration:

```
AT_ERRCODE ACCUAPI IG_cpm_image_embedded_profile_check(
        HIGEAR hIGear,
        LPAT_BOOL lpbEmbedded
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	(in) A handle of the image to check.
lpbEmbedded	LPAT_BOOL	(out) Return TRUE if profile embedded, otherwise, FALSE.

Return Value:

Return value is a code of last error or NULL if success.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;  /* Handle of the image to check*/
AT_BOOL bEmbeddedProfile = FALSE;       /* Flag to return whether profile embedded or not */
/* Load an image into hIGear */
......
/* Check whether the image has embedded profile. */
if (IGE_SUCCESS == IG_cpm_image_embedded_profile_check(hIGear, &bEmbeddedProfile))
{
......
}
```

1.3.1.2.6.2 IG_cpm_image_profile_get

This function returns information about color profile associated with given image.

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_image_profile_get(
        HIGEAR hIGear,
        LPAT_BOOL lpbIsLocal,
        LPAT_MODE lpnColorSpace,
        LPCHAR lpStatusStr,
        UINT nStatusSize
        LPUINT lpnStatusLen,
        LPDWORD lpnProfileSize
        LPBYTE lpProfileData,
        DWORD dwProfileDataSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	IN: handle of image where to set profile.
lpbIsLocal	LPAT_BOOL	OUT: return TRUE if local profile associated with given image and FALSE in other case. If NULL then argument ignored.
lpnColorSpace	LPAT_MODE	OUT: color space id of returned profile. Possible returned values: IG_COLOR_SPACE_RGBIG_COLOR_SPACE_CMYK. If NULL then parameter ignored.
lpStatusStr	LPCHAR	OUT: pointer where to copy textual information about local profile. If NULL then parameter ignored.
nStatusSize	UINT	IN: size of lpStatusStr buffer.
lpnStatusLen	LPUINT	OUT: Return length of actual status message. If NULL then parameter ignored.
lpnProfileSize	LPDWORD	OUT: If not NULL then returns actual profile size in ICC format.
lpProfileData	LPBYTE	OUT: Pointer where to put profile data. Profile will be written according to ICC.1:1998-09 specification.
dwProfileDataSize	DWORD	IN: the size of lpProfileData buffer.

Return Value:

Return value is the code of the last error, or NULL if success.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

The second parameter is used to get information either this image has embedded local profile or use global profile.

The rest of parameters are used to provide text description of profile and profile data itself. The profile returned through lpProfileData will be written according to ICC specification.

Depending from parameters it either return information about profile or profile data itself. For example, if lpnProfileSize is not NULL then the size of profile is calculated and returned. If lpProfileData is not NULL then profile is encoded into standard ICC format and returned through this parameter.

Please note that color profile management is disabled by default. See Working with ImageGear Color Profile Manager for a description of how to activate it.

1.3.1.2.6.3 IG_cpm_image_profile_set

This function provides color profile management for given particular image.

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_image_profile_set(
        HIGEAR hIGear,
        LPAT_BYTE lpRawData,
        DWORD dwRawSize,
        AT_BOOL bConvert
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	IN: handle of image where to set profile.
lpRawData	LPAT_BYTE	IN: raw data of new profile. Can be either NULL or pointer to memory buffer that contains valid color profile in format specified by ICC.1:1998-09.
dwRawSize	DWORD	IN: length of data stored in lpRawData.
bConvert	AT_BOOL	IN: if TRUE then convert all images associated with old profile to new profile, but if FALSE then simple replace profile without any conversion.

Return Value:

Return value is a code of last error, or NULL if success.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

There can be two ways that an image is associated with a color profile:

- First way: image itself does not store color profile data but use profile from global parameters (WCP).
- Second way: image stores color profile locally and does not depend on global settings.

If lpRawData is NULL during this function call, then previous local profile (if it existed) is deleted and image become dependent on global profile correspondent to color space used by its pixel data. If lpRawData is valid ICC profile then previous local profile (if it existed) is deleted and image becomes associated with new local profile.

The last parameter specifies how pixel data should be changed during profile change operation. If bConvert is TRUE, then pixel is converted from one format to another but in other case pixel data is unchanged.

Please note, that color profile management is disabled by default. See Working with ImageGear Color Profile Manager for information about how to activate it.

If some error happens such as invalid or unsupported profile or color space mismatch in image and color profile, then function returns appropriate error code.

1.3.1.2.6.4 IG_cpm_profile_get

This function returns information about global profile for given color space of given group.

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_profile_get(
        AT_MODE nColorSpace,
        DWORD nProfileGroup,
        LPCHAR lpStatusStr,
        UINT nStatusSize
        LPUINT lpnStatusLen,
        LPDWORD lpnProfileSize,
        LPBYTE lpProfileData,
        DWORD dwProfileDataSize
);
```

Arguments:

Name	Type	Description
nColorSpace	AT_MODE	IN: color space ID which profile status to get. Possible value: IG_COLOR_SPACE_RGBIG_COLOR_SPACE_CMYK.
nProfileGroup	DWORD	IN: color profile group where profiles to get. Possible value: IG_CP_GRP_DISPLAYIG_CP_GRP_WORKINGIG_CP_GRP_EXPORTIG_CP_GRP_IMPORT.
lpStatusStr	LPCHAR	OUT: pointer where to copy textual information about local profile. If NULL then parameter is ignored.
nStatusSize	UINT	IN: size of lpStatusStr buffer.
lpnStatusLen	LPUINT	OUT: return length of actual status message. If NULL then parameter ignored.
lpnProfileSize	LPDWORD	OUT: If this parameter is not NULL then actual size of color profile in ICC format is returned.
lpProfileData	LPBYTE	OUT: Pointer where to put profile data or ignored if parameter is NULL. Profile will be written according to ICC.1:1998-09 specification.
dwProfileDataSize	DWORD	IN: the size of lpProfileData.

Return Value:

Return value is a code of last error or 0 if success.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Depending from parameters it either returns information about profile or profile data itself. For example, if lpnProfileSize is not NULL, then size of profile is calculated and returned. If lpProfileData is not NULL, then profile is encoded into standard ICC format and returned through this parameter.

Please note that color profile management is disabled by default. See section Working with ImageGear Color Profile Manager for information about how to activate it.

1.3.1.2.6.5 IG_cpm_profile_set

This function sets new value of color profile associated with color space given by nColorSpace parameter to profile group given by nProfileGroup parameter.

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_profile_set(
        AT_MODE nColorSpace,
        DWORD nProfileGroup
        LPAT_BYTE lpRawData,
        DWORD dwRawSize,
        AT_BOOL bConvert
);
```

Arguments:

Name	Type	Description
nColorSpace	AT_MODE	IN: color space ID which profile to replace. Current supported values: IG_COLOR_SPACE_RGBIG_COLOR_SPACE_CMYK.
nProfileGroup	DWORD	IN: color profile group where to set profile. Possible values: IG_CP_GRP_WORKINGIG_CP_GRP_IMPORTIG_CP_GRP_EXPORTIG_CP_GRP_DISPLAY.
lpRawData	LPAT_BYTE	IN: raw data of new profile. Can be either NULL or pointer to memory buffer that contains valid color profile in format specified by ICC.1:1998-09.
dwRawSize	DWORD	IN: length of data stored in lpRawData.
bConvert	AT_BOOL	IN: if TRUE then this function converts all images associated with old profile to new profile, but if FALSE it simply replaces profile without any conversion.

Return Value:

Return value is a code of last error or 0 if success.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If the last parameter is set to TRUE, then all associated images with previous format are converted to new color profile. In the case that nColorSpace does not match the same color space that is used in color profile or color profile is in invalid or unsupported format, then error is returned.

Please note that color profile management is disabled by default. See Working with ImageGear Color Profile Manager for information about how to activate it.

1.3.1.2.6.6 IG_cpm_profiles_reset

This function resets all default color profiles to default values taken from global parameters.

Declaration:

```
AT_ERRCODE LACCUAPI IG_cpm_profiles_reset(
        AT_BOOL bConvert
);
```

Arguments:

Name	Type	Description
bConvert	AT_BOOL	IN: specify how this operation should affect associated image.

Return Value:

Return value is a code of last error or 0 if success.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

If value of parameter is TRUE then it converts all images associated with all global profiles to default profiles, but if FALSE then all global profiles are reset to default values but images are not changed.

Please note that color profile management is disabled by default. See Using Color Profile Manager for information about how to activate it.

1.3.1.2.7 DIB Functions

This section provides information about the DIB group of functions.

- IG_DIB_area_get
- IG_DIB_area_set
- IG_DIB_area_size_get
- IG_DIB_bit_depth_get
- IG_DIB_channel_count_get
- IG_DIB_channel_depth_get
- IG_DIB_channel_depths_get
- IG_DIB_colorspace_get
- IG_DIB_column_get
- IG_DIB_column_set
- IG_DIB_flood_fill
- IG_DIB_flush
- IG_DIB_height_get
- IG_DIB_info_copy
- IG_DIB_info_create
- IG_DIB_info_delete
- IG_DIB_info_raster_size_get
- IG_DIB_legacy_bit_depth_get
- IG_DIB_line_get
- IG_DIB_line_set
- IG_DIB_palette_alloc
- IG_DIB_palette_length_get
- IG_DIB_palette_pointer_get
- IG_DIB_palette_size_get
- IG_DIB_pixel_array_size_get
- IG_DIB_pixel_get
- IG_DIB_pixel_set
- IG_DIB_pix_get
- IG_DIB_pix_set
- IG_DIB_raster_get
- IG_DIB_raster_set
- IG_DIB_raster_size_get
- IG_DIB_resolution_get
- IG_DIB_resolution_set
- IG_DIB_resolution_units_get
- IG_DIB_resolution_units_set
- IG_DIB_row_get
- IG_DIB_row_set
- IG_DIB_width_get

1.3.1.2.7.1 IG_DIB_area_get

This function obtains all the pixels contained within the rectangular portion of image hIGear specified by lpRect.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_area_get (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        LPAT_PIXEL lpPixel,
        AT_MODE nPixelFormat
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	The hIGear handle of an image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image bitmap to get.
lpPixel	LPAT_PIXEL	Far pointer to first in an array of bytes large enough to receive all pixels in the area.
nPixelFormat	AT_MODE	A constant such as IG_DIB_AREA_UNPACKED, specifying in what form you want the pixels stored in your array. The list of IG_DIB_AREA_ constants available is in file accucnst.h.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR          hIGear;       /* HIGEAR handle of image  */
AT_RECT         rcBlock;      /* The rectangular block to get  */
AT_PIXEL        cPixArray[400];/* Will receive returned pixels  */
AT_DIMENSION    nWid, nHi;     /* Will receive width, height of image     */
UINT            nBpp;          /* Bits per pixel  */
AT_ERRCOUNT    nErrcount;    /* Will receive returned error counts      */
/* Will fetch upper left 20 x 20 pixels, to cPixArray[]:   */
rcBlock.top = rcBlock.left = 0;
rcBlock.bottom = rcBlock.right = 20;  /* 20x20 area, 400 pixels  */
nErrcount = IG_image_dimensions_get ( hIGear, &nWid, &nHi, &nBpp );  */
if ( nErrcount == 0 )       /* If valid image, dimensions obtained'    */
        {
        if ( (nBpp <= 8) && (nWid >= 20) && (nHi >= 20) )
               {/* (Array is too small for 24-bit)                            */
               nErrcount = IG_DIB_area_get ( hIGear, &rcBlock,
                       &cPixelArray[0], IG_DIB_AREA_UNPACKED );
               }
        }
```

Remarks:

Use the lpPixel argument to tell ImageGear where to store the pixels.

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

Use nPixelFormat = IG_DIB_AREA_DIB if you want the data in standard uncompressed DIB format, and with each row returned to you padded to a multiple of 4 bytes length. 1-bit pixels are returned 8 to the byte, most significant bit first. 4-bit pixels are returned 2 to the byte, similarly left justified. 24-bit pixels are returned 3 bytes per pixel, ordered Blue-Green-Red.

Use nPixelFormat = IG_DIB_AREA_UNPACKED if you want the pixels returned 1 per byte (but still 3 bytes for a 24-bit pixel, ordered Blue-Green-Red). Each 1-bit or 4-bit pixel will be returned right justified in a single byte, padded with zeroes in the most significant bits of the byte.

In either case, be sure your area pointed to by lpPixel is large enough to receive all the pixel data including padding.

See also function IG_DIB_area_set().

1.3.1.2.7.2 IG_DIB_area_set

This function transfers pixels from the location pointed to by lpPixel into the rectangular portion of the image specified by rectangle lpRect.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_area_set (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const LPAT_PIXEL lpPixel,
        AT_MODE nPixelFormat
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear HIGEAR handle of image
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image bitmap to set
lpPixel	LPAT_PIXEL	Far pointer to first byte of your pixel data
nPixelFormat	AT_MODE	A constant such as IG_DIB_AREA_UNPACKED specifying in what form you are providing the pixels. The IG_DIB_AREA_... constants are listed in file accucnst.h.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR          hIGear;        /* HIGEAR handle of image */
AT_RECT         rcBlock;       /* The rectangular block to set    */
AT_PIXEL        cPixelArray[400];    /* The pixels to set */
AT_DIMENSION    nWid, nHi;     /* Receives the width & height of an image */
UINT            nBpp;          /* Bits per pixel    */
AT_ERRCOUNT     nErrcount;      /* Receives the returned error counts   */
/* Sets the upper left 20 x 20 pixels, to cPixArray[]:                 */
rcBlock.top = rcBlock.left = 0;
rcBlock.bottom = rcBlock.right = 20;  /* 20x20 area, 400 pixels    */
nErrcount = IG_image_dimensions_get ( hIGear, &nWid, &nHi, &nBpp );   */
if ( nErrcount == 0 )       /* If valid image, dimensions obtained:   */
        {
        if ( (nBpp <= 8) && (nWid >= 20) && (nHi >= 20) )
{
/*Array is too small for 24-bit)      */
        INT           row, col; pix;     /* For the loops below   */
        AT_PIXEL      nPixval;          /* pixel value to set   */
        if (nBpp == 8) nPixval = 128;   /* Value to set if 8-bit    */
        if (nBpp == 4) nPixval = 8;     /* Value to set if 4-bit    */
        if (nBpp == 1) nPixval = 1;     /* Pixel ON if 1-bit     */
                for ( pix=0,row=0; row<20; row++ ) /* For all pixels in     */
                        for ( col=0; col<20; col++ )/* the 20 x 20 array:   */
                                cPixelArray[pix++] = nPixval;
                                /* Set unpacked in byte     */
                nErrcount = IG_DIB_area_set ( hIGear, &rcBlock, &cPixelArray[0],
IG_DIB_AREA_UNPACKED );
}
        }
```

Remarks:

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

- Use nPixelFormat = IG_DIB_AREA_DIB if you are providing the pixels in standard uncompressed DIB format. This means 1-bit or 4-bit pixels will be packed 8 to the byte or 2 to the byte respectively, left justified (first pixel uses most significant bit). 24-bit pixels are in 3 bytes, ordered Blue-Green-Red.
- Use nPixelFormat = IG_DIB_AREA_UNPACKED if you are providing the pixels 1 pixel per byte (however, 3 bytes for a 24-bit pixel, ordered Blue-Green-Red). In this case, you provide 1-bit and 4-bit pixels one to a byte, right justified in the byte.

See also function IG_DIB_area_get().

1.3.1.2.7.3 IG_DIB_area_size_get

This function calculates and returns the number of bytes required to hold a rectangular region selected from an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_area_size_get(
        HIGEAR hIGear,
        LPAT_RECT lpRect,
        AT_MODE nFormat,
        LPAT_DIMENSION lpSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	LPAT_RECT	A far pointer to a rectangular array of pixels from the image. Setting this to NULL selects the whole image.
nFormat	AT_MODE	A variable of type AT_MODE, such as IG_DIB_AREA_DIB, that defines how the data should be stored: packed or unpacked. (They are defined in accucnst.h)
lpSize	LPAT_DIMENSION	A far pointer to a variable that returns the size in bytes of the array of pixels.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

You can use this function to determine the size of the block of memory to allocate to hold the pixel values from the rectangular region. This will help you to avoid data overflow. The value returned by lpSize includes the allocation of space for raster buffering at the end of each raster line. (See the section Device-Independent Bitmaps (DIB)Understanding Bitmap Images.)

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

Use the format parameter to indicate the form in which you want to store the pixels:

Use:	To:
IG_DIB_AREA_DIB	Pad rows to long boundaries (the way they are stored in a DIB).
IG_DIB_AREA_UNPACKED	Store pixels 1 per byte for 1, 4, 8-bit images, 1 per 3 bytes for a 24-bit image.

1.3.1.2.7.4 IG_DIB_bit_depth_get

This function returns the bit depth of an image, which is the sum of the channel bit depths.

Declaration:

```
AT_INT ACCUAPI IG_DIB_bit_depth_get(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Returns image bit depth.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT imageDepth;      /* Returned bit depth of image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
imageDepth = IG_DIB_bit_depth_get(hDIBInfo);
```

Remarks:

For example, if the image is a simple 24-bit RGB image, this function will return 24. If it's a 24-bit RGB image with an 8-bit alpha channel, this function will return 32.

1.3.1.2.7.5 IG_DIB_channel_count_get

This function returns the number of channels in the image.

Declaration:

```
AT_INT ACCUAPI IG_DIB_channel_count_get(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	**Type**	**Description**
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Returns the number of channels in the image.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT nChannels;       /* Returned number of channels */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
nChannels = IG_DIB_channel_count_get(hDIBInfo);
```

Remarks:

For example, a typical 24-bit RGB image has three 8-bit channels (red, green, blue) so this function will return 3. For a 24-bit RGB image with a single alpha channel, this function would return 4.

1.3.1.2.7.6 IG_DIB_channel_depth_get

This function returns the bit depth of the channel specified by the Index parameter.

Declaration:

```
AT_INT ACCUAPI IG_DIB_channel_depth_get(
        HIGDIBINFO hDIB,
        AT_INT Index
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.
Index	AT_INT	Index of channel of which to return bit depth.

Return Value:

Returns specified channel's bit depth.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depth;           /* Returned depth of first channel */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
depth = IG_DIB_channel_depth_get(hDIBInfo, 0);
```

Remarks:

For example, if you had a typical 24-bit RGB image with three 8-bit channels, you could specify 0, 1, or 2 for the index and the return value would be 8.

1.3.1.2.7.7 IG_DIB_channel_depths_get

This function copies channel bit depths to an array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_channel_depths_get(
        HIGDIBINFO hDIB,
        AT_INT ChannelCount,
        AT_INT* lpChannelDepths
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.
ChannelCount	AT_INT	Number of channels for which to return depths.
lpChannelDepths	AT_INT*	Array to which to copy channel depths.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT nChannels;       /* Number of channels in image */
LPAT_INT depths;        /* Array of channel depths */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
nChannels = IG_DIB_channel_count_get(hDIBInfo);
depths = (LPAT_INT) malloc(nChannels * sizeof(AT_INT));
nErrcount = IG_DIB_channel_depths_get(hDIBInfo, nChannels, depths);
```

Remarks:

You can use this function to find out the bit depths of each individual channel.

1.3.1.2.7.8 IG_DIB_colorspace_get

This function returns the image's color space.

Declaration:

```
enumIGColorSpaceIDs ACCUAPI IG_DIB_colorspace_get(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Returns a combination of values from enumIGColorSpaceIDs.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
enumIGColorSpaceIDs colorspace; /* Color space of image */
AT_BOOL bIndexed;       /* Is color space of image indexed? */
AT_BOOL bHasAlpha;      /* Does image have an alpha channel? */
/* Find out if an image is indexed, and if it has alpha */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
colorspace = IG_DIB_colorspace_get(hDIBInfo);
bIndexed = (colorspace & IG_COLOR_SPACE_ID_ColorMask) == IG_COLOR_SPACE_ID_I;
bHasAlpha = colorspace & IG_COLOR_SPACE_ID_A;
```

Remarks:

This is a bitmask that indicates the color space as well as the presence of alpha, pre-multiplied alpha, and extra channels.

See the IG_util_colorspace_...() functions for more functions that retrieve information about color spaces, alpha, and extra channels.

Use caution when extracting information from this bitmask. For example, if you want to look at only the color space, you must use the mask IG_COLOR_SPACE_ID_ColorMask as in the example.

1.3.1.2.7.9 IG_DIB_column_get

This function obtains a column of pixels from the DIB image bitmap of image hIGear, and stores the pixels in your data area pointed to by lpPixel.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_column_get (
        HIGEAR hIGear,
        AT_PIXPOS nX,
        AT_PIXPOS nY1,
        AT_PIXPOS nY2,
        LPAT_PIXEL lpPixel,
        AT_DIMENSION nLenBytes,
        LPAT_DIMENSION lpNumPixels
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nX	AT_PIXPOS	X offset (X coord) of the vertical pixel column to get.
nY1	AT_PIXPOS	Raster line number at which the vertical column starts.
nY2	AT_PIXPOS	Raster line number at which the vertical column ends.
lpPixel	LPAT_PIXEL	Far pointer to your data area to which the pixels should be returned.
nLenBytes	AT_DIMENSION	Size of the data area, in bytes.
lpNumPixels	LPAT_DIMENSION	Far pointer to a AT_DIMENSION variable in which will be returned the number of pixels (not bytes) transferred.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;            /* HIGEAR handle of image
*/
AT_PIXEL cPixArray[400];  /* Receives the returned pixels       */
AT_PIXPOS nCol, nYtop, nYbot; /* Column and which rows to get           */
AT_DIMENSION nFetched;  /* Holds count of pixels retrieved      */
AT_ERRCOUNT nErrcount;  /* Receives returned error counts       */
nCol  = 0;   /* Fetch left boundary of image    */
nYtop = 10;  nYbot = 59;   /* 50 pixels, from lines 10 thru 59          */
nErrcount = IG_DIB_column_get ( hIGear, nCol, nYtop, nYbot, &cPixArray[0], 400, &nFetched
);
```

Remarks:

The offset of the column to retrieve is indicated by nX. The beginning and ending raster lines of the column are nY1 and nY2, respectively (top to bottom).

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

If the pixels will not fit in nLenBytes, only nLenBytes of data will be transferred. The actual number of pixels transferred is returned in the variable pointed to by lpNumPixels.

If the image is 1-bit or 4-bit, the pixels will be returned one to a byte, right justified in the byte. If the image is 24-bit, each pixel will occupy 3 bytes, in Blue-Green-Red order.

1.3.1.2.7.10 IG_DIB_column_set

This function sets the pixel data you supply into the DIB image bitmap column specified by nX.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_column_set (
        HIGEAR hIGear,
        AT_PIXPOS nX,
        AT_PIXPOS nY1,
        AT_PIXPOS nY2,
        const LPAT_PIXEL lpPixel,
        AT_DIMENSION nNumPixels
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nX	AT_PIXPOS	X offset (X coord) of the vertical pixel column to be set.
nY1	AT_PIXPOS	Raster line number at which the vertical column starts.
nY2	AT_PIXPOS	Raster line number at which the column ends.
lpPixel	const LPAT_PIXEL	Far pointer to first byte of your pixel data.
nNumPixels	AT_DIMENSION	Number of pixels (not bytes) to transfer.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of image            */
AT_PIXEL cPixArray[400];  /* Receives the returned pixels      */
AT_PIXPOS nCol,  nYtop,  nYbot; /* Column and rows to get          */
AT_DIMENSION nFetched;  /* Holds the count of pixels retrieved*/
AT_ERRCOUNT nErrcount;  /* Receives the returned error counts */
/* Restore the pixels saved by the call in example IG_DIB_column_get:   */
nCol  = 0;   /* Restores to image's left boundary */
nYtop = 10;  nYbot = 59;   /* 50 pixels, from lines 10 thru 59*/
nFetched = nYbot - Top + 1;
nErrcount = IG_DIB_column_set ( hIGear, nCol, nYtop, nYbot, &cPixArray[0],nFetched );
```

Remarks:

The pixel at nX is set in pixel rows nY1 through nY2 inclusive. nNumPixels is the number of pixels to set, and should equal (nY2-nY1+1).

> ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

If the image is 1-bit or 4-bit, your pixels should be one to a byte and right justified, beginning at lpPixel. If the image is 24-bit, each pixel should occupy 3 bytes, in Blue-Green-Red order.

If the image you are modifying is 1-bit, you will probably need to convert the image from run-end encoded to a standard DIB before you can set pixel values. Please see the section Accessing Image Pixels for details.

1.3.1.2.7.11 IG_DIB_flood_fill

This function fills an area with a color, starting at the specified point.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_flood_fill(
        HIGEAR hIGear,
        AT_INT xPos,
        AT_INT yPos,
        LPAT_PIXEL lpFillPixel,
        LPAT_PIXEL lpBorderPixel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image to process.
xPos	AT_INT	X coordinate of a point inside the area to be filled.
yPos	AT_INT	X coordinate of a point inside the area to be filled.
lpFillPixel	LPAT_PIXEL	Fill color.
lpBorderPixel	LPAT_PIXEL	Color of the border surrounding the area.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB – 1 bpp;
Grayscale – 1 bpp.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Remarks:

There are two ways to specify the area:

- If BorderColor is NULL, the area is defined by the color of the specified point.
- Otherwise, the area is defined by the border color.

1.3.1.2.7.12 IG_DIB_flush

This function flushes the DIB if it uses a memory mapped file for storing pixels.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_flush(HIGEAR hIGear);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Handle of the image whose pixel data should be flushed.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB - 1 bpp;
- Grayscale - 1 bpp.

Remarks:

Call this function periodically to flush the DIB if the application accesses individual pixels or rasters of an image, or accesses pixel data directly by the image or raster pointer, and the DIB uses a memory mapped file for storing pixels. Usually, it is sufficient to flush the DIB after accessing of 100 - 200 Mb of pixel data.

If memory mapping is not used for the DIB, the function does nothing.

See Also:

Accessing Pixels of a Gigabyte-Sized Image

1.3.1.2.7.13 IG_DIB_height_get

This function returns the height of the image.

Declaration:

```
AT_DIMENSION ACCUAPI IG_DIB_height_get(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Returns image height.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_DIMENSION height;    /* Returned height of image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
height = IG_DIB_height_get(hDIBInfo);
```

1.3.1.2.7.14 IG_DIB_info_copy

This function makes a copy of a DIB info object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_info_copy(
        HIGDIBINFO hDIBSrc,
        HIGDIBINFO* lphDIBDst
);
```

Arguments:

Name	Type	Description
hDIBSrc	HIGDIBINFO	DIB info handle from which to copy.
lphDIBDst	HIGDIBINFO*	Pointer to where copied DIB info handle will be stored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;   /* Number of errors on stack */
HIGDIBINFO hDIBInfo;     /* Handle of DIB info to be copied */
HIGDIBINFO hDIBInfoCopy; /* Handle of DIB info copy */
HIGEAR hImage;           /* HIGEAR handle of image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
nErrcount = IG_DIB_info_copy(hDIBInfo, &hDIBInfoCopy);
```

Remarks:

This function does not copy pixel data.

1.3.1.2.7.15 IG_DIB_info_create

This function creates a new DIB info object and initializes it to the given values.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_info_create(
        HIGDIBINFO* lphDIB,
        AT_DIMENSION width,
        AT_DIMENSION height,
        enumIGColorSpaceIDs colorspace,
        AT_INT channelCount,
        AT_INT* channelDepths
);
```

Arguments:

Name	Type	Description
lphDIB	HIGDIBINFO*	Pointer to where created DIB info's handle will be stored.
width	AT_DIMENSION	Width of image in pixels.
height	AT_DIMENSION	Height of image in pixels.
colorspace	enumIGColorSpaceIDs	Color space of image.
channelCount	AT_INT	Number of channels in image.
channelDepths	AT_INT*	Array of channel depths.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
/* Create a DIB info object describing a 48-bit RGB image */
AT_ERRCOUNT nErrcount;   /* Number of errors on stack */
HIGDIBINFO hDIBInfo;             /* DIB info handle */
AT_INT depths[3] = {16, 16, 16}; /* Array of channel depths */
nErrcount = IG_DIB_info_create(&hDIBInfo, 320, 240, IG_COLOR_SPACE_ID_RGB, 3, depths);
/* ... */
nErrcount = IG_DIB_info_delete(hDIBInfo);
```

Remarks:

The DIB info object must be deleted with IG_DIB_info_delete when it is finished being used.

This function does not allocate pixel data storage.

1.3.1.2.7.16 IG_DIB_info_delete

This function deletes a DIB info object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_info_delete(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
/* Create a DIB info object describing a 24-bit RGB image */
AT_ERRCOUNT nErrcount;   /* Number of errors on stack */
HIGDIBINFO hDIBInfo;             /* DIB info handle */
AT_INT depths[3] = {8, 8, 8};    /* Array of channel depths */
nErrcount = IG_DIB_info_create(&hDIBInfo, 320, 240, IG_COLOR_SPACE_ID_RGB, 3, depths);
/* ... */
nErrcount = IG_DIB_info_delete(hDIBInfo);
```

1.3.1.2.7.17 IG_DIB_info_raster_size_get

This function returns the number of bytes per raster in the DIB.

Declaration:

```
AT_INT ACCUAPI IG_DIB_info_raster_size_get(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Returns DIB raster size.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGDIBINFO hDIBInfo;    /* DIB info handle */
AT_INT rasterSize;      /* Returned raster size */
rasterSize = IG_DIB_info_raster_size_get(hDIBInfo);
```

Remarks:

For all images except 1-bit images, the returned number is precise.

For 1-bit images, the function returns the raster size assuming that 8 pixels are packed per byte. In reality, ImageGear internally stores 1-bit images using run ends scheme. The size of the raster stored in memory depends on the content of the raster, but usually the raster occupies much less space than the number returned by IG_DIB_info_raster_size_get.

1.3.1.2.7.18 IG_DIB_legacy_bit_depth_get

This function returns the bit depth that earlier ImageGear versions used to store this image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_legacy_bit_depth_get(
        HIGEAR hIGear,
        LPAT_INT lpBitsPerPixel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpBitsPerPixel	LPAT_INT	Returned legacy bit depth of the image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
AT_INT bpp;             /* Image bit depth */
nErrcount = IG_DIB_legacy_bit_depth_get(hImage, &bpp);
/* bpp could be 1, 4, 8, 9-16, 24, or 32 */
```

Remarks:

This function can be used for working with pixel access functions in the legacy mode.

1.3.1.2.7.19 IG_DIB_line_get

This function obtains an arbitrary line of pixels from the DIB image bitmap of the image referenced by hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_line_get (
        HIGEAR hIGear,
        AT_PIXPOS nX1,
        AT_PIXPOS nY1,
        AT_PIXPOS nX2,
        AT_PIXPOS nY2,
        LPAT_PIXEL lpPixel,
        const AT_DIMENSION nlenOfArray,
        LPAT_DIMENSION lpNumPixels
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nX1	AT_PIXPOS	X coordinate of the first endpoint of line to get.
nY1	AT_PIXPOS	Y coordinate of the first endpoint.
nX2	AT_PIXPOS	X coordinate of the second endpoint of line.
nY2	AT_PIXPOS	Y coordinate of the second endpoint.
lpPixel	LPAT_PIXEL	Far pointer to your data area to which the pixels should be returned.
nLenOfArray	const AT_DIMENSION	Length, in bytes, of lpPixel block.
lpNumPixels	LPAT_DIMENSION	Far pointer to an AT_DIMENSION variable in which the number of pixels (not bytes) transferred is returned.

Return Value:

Returns the number of ImageGear errors that occurred during the function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

See the example for function IG_DIB_line_set().

Remarks:

The line of pixels does not have to be horizontal or vertical. nX1,nY1 are the coordinates of one endpoint of the line and nX2,nY2 are of the other.

If the pixel data would overflow your area (nLenOfArray), the transfer will be truncated; your data area will not be overflowed. The actual number of pixels returned will be stored in your variable pointed to by lpNumPixels.

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

To determine the number of pixels that your line will be comprised of, use the following formula:
max((1+abs(x2-x1)), (1+abs(y2-y1)))
1-bit and 4-bit pixels are returned one to a byte, right justified. 24-bit pixels are returned 3 bytes per pixel, in Blue-Green-Red order.

1.3.1.2.7.20 IG_DIB_line_set

This function stores an arbitrary line of pixels in the DIB image bitmap of the image referenced by HIGEAR.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_line_set (
        HIGEAR hIGear,
        AT_PIXPOS nX1,
        AT_PIXPOS nY1,
        AT_PIXPOS nX2,
        AT_PIXPOS nY2,
        const LPAT_PIXEL lpPixel,
        AT_DIMENSION nNumPixels
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to which to transfer pixels.
nX1	AT_PIXPOS	X coordinate of the first endpoint of line to store.
nY1	AT_PIXPOS	Y coordinate of the first endpoint.
nX2	AT_PIXPOS	X coordinate of the second endpoint of line to store.
nY2	AT_PIXPOS	Y coordinate of the second endpoint.
lpPixel	const LPAT_PIXEL	Far pointer to your data area containing the pixels to be stored.
nNumPixels	AT_DIMENSION	Number of pixels (not bytes) to be transferred.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;       /* HIGEAR handle of image              */
AT_PIXEL cPixArray[400]; /* Receives the returned pixels         */
AT_PIXPOS nXleft, nYtop,  /* Coordinates of upper-left end of line*/
  nXright, nYbot; /* Coordinates of the lower-right end   */
AT_DIMENSION nFetched;     /* Holds the count of pixels retrieved  */
AT_ERRCOUNT  nErrcount;            /* Receives the returned error counts   */
/* Fetch a diagonal line, from Coordinates (0,0) to (100,100):
*/
nXleft  = 0;    nYtop = 0;                            /* Diagonal line from
upper left corner      */
nXright = 100;  nYbot = 100;
nErrcount = IG_DIB_line_get ( hIGear, nXleft, nYtop, nXright, nYbot, &cPixArray[0], 400,
&nFetched );
 ...
/* Now restore the line: */
nErrcount = IG_DIB_line_set ( hIGear, nXleft, nYtop, nXright, nYbot, &cPixArray[0],
nFetched );
```

Remarks:

The line does not have to be horizontal or vertical. (nX1,nY1) are the coordinates of one endpoint of the line and (nX2,nY2) are of the other.

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

If the image you are modifying is 1-bit, you must convert the image from run-end encoded to a standard DIB before you can set pixel values. Please see the section Accessing Image Pixels for details.

1-bit and 4-bit pixels should be provided one to a byte, right justified (that is, in the least significant bits of the byte). 24 bit pixels should be 3 bytes per pixel, in Blue-Green-Red order.

To calculate the number of pixels that your line will consist of, use the following formula:
max((1+abs(x2-x1)), (1+abs(y2-y1)))

1.3.1.2.7.21 IG_DIB_palette_alloc

This function allocates a palette for the given DIB.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_palette_alloc(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB: 1…8 bpp

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_INT nEntries;        /* Number of palette entries */
LPAT_RGBQUAD lpPalette; /* Pointer to palette data */
AT_INT i;               /* Index for palette loop */
/* Make a palette in which every color is GREEN */
nErrcount = IG_DIB_palette_alloc(hDIB);
nEntries = IG_DIB_palette_length_get(hDIB);
lpPalette = IG_DIB_palette_pointer_get(hDIB);
for (i = 0; i < nEntries; i++)
{
        lpPalette[i].rgbRed = lpPalette[i].rgbBlue = 0;
        lpPalette[i].rgbGreen = 255;
}
```

1.3.1.2.7.22 IG_DIB_palette_length_get

This function returns the number of entries in the DIB's palette.

Declaration:

```
AT_INT ACCUAPI IG_DIB_palette_length_get(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Returns the number of palette entries.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_INT nEntries;        /* Number of palette entries */
LPAT_RGBQUAD lpPalette; /* Pointer to palette data */
AT_INT i;               /* Index for palette loop */
/* Make a palette in which every color is GREEN */
nErrcount = IG_DIB_palette_alloc(hDIB);
nEntries = IG_DIB_palette_length_get(hDIB);
lpPalette = IG_DIB_palette_pointer_get(hDIB);
for (i = 0; i < nEntries; i++)
{
        lpPalette[i].rgbRed = lpPalette[i].rgbBlue = 0;
        lpPalette[i].rgbGreen = 255;
}
```

1.3.1.2.7.23 IG_DIB_palette_pointer_get

This function returns a pointer to the DIB's palette, if it is present; otherwise it returns NULL.

Declaration:

```
LPAT_RGBQUAD ACCUAPI IG_DIB_palette_pointer_get(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Pointer to the DIB palette, if it is present; NULL - otherwise.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_INT nEntries;        /* Number of palette entries */
LPAT_RGBQUAD lpPalette; /* Pointer to palette data */
AT_INT i;               /* Index for palette loop */
/* Make a palette in which every color is GREEN */
nErrcount = IG_DIB_palette_alloc(hDIB);
nEntries = IG_DIB_palette_length_get(hDIB);
lpPalette = IG_DIB_palette_pointer_get(hDIB);
for (i = 0; i < nEntries; i++)
{
        lpPalette[i].rgbRed = lpPalette[i].rgbBlue = 0;
        lpPalette[i].rgbGreen = 255;
}
```

Remarks:

You can use this to get and set palette entries or the palette as a whole.

1.3.1.2.7.24 IG_DIB_palette_size_get

This function returns the size of the DIB's palette, in bytes.

Declaration:

```
AT_INT ACCUAPI IG_DIB_palette_size_get(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

DIB palette size, in bytes.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGDIBINFO hDIB;        /* DIB info handle */
AT_INT paletteSize;     /* Returned size of palette */
paletteSize = IG_DIB_palette_size_get(hDIB);
```

1.3.1.2.7.25 IG_DIB_pixel_array_size_get

This function returns the number of bytes needed to store an array of pixels.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pixel_array_size_get(
        HIGEAR hIGear,
        AT_DIMENSION length,
        AT_MODE format,
        LPAT_DIMENSION lpSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image containing pixel data.
length	AT_DIMENSION	Number of pixels for which to calculate array size.
format	AT_MODE	IG_PIXEL_UNPACKED - All bit depths are unpacked (at least one byte per pixel). IG_PIXEL_PACKED - In legacy mode: 1 and 4 bit images are packed (8 or 2 pixels per byte). In new mode: Only 1 bit images are packed (8 pixels per byte).
lpSize	LPAT_DIMENSION	Returned array size in bytes.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
/* Get # of bytes needed to store half a row */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
AT_DIMENSION w, h;      /* Width and height of image */
AT_DIMENSION nBytes;    /* Size of pixel array in bytes */
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
nErrcount = IG_DIB_pixel_array_size_get(hImage, w / 2,
    IG_PIXEL_UNPACKED, &nBytes);
```

Remarks:

This can be used to allocate storage for use with pixel access functions. This function is similar to IG_DIB_raster_size_get(), but it lets you specify the number of pixels instead of using the number of pixels in an entire raster.

1.3.1.2.7.26 IG_DIB_pixel_get

This function obtains the pixel at coordinates (nXpos, nYpos), storing it right justified (that is, in the least significant bits) at the location pointed to by lpPixel.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pixel_get (
        HIGEAR hIGear,
        AT_PIXPOS nXpos,
        AT_PIXPOS nYpos,
        LPAT_PIXEL lpPixel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nXpos	AT_PIXPOS	X offset (in pixels) from beginning of raster line. First pixel on line is pixel number 0.
nYpos	AT_PIXPOS	Raster line number. 0 is top line.
lpPixel	LPAT_PIXEL	Far pointer to byte at which to store pixel (or to a 3-byte area if a 24-bit pixel).

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Pixel Access, FlashPix

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image                                          */
AT_PIXEL                cPixelValue[3];/* 3 bytes in case 24-bit image
*/
                /* Get value of the upper-leftmost pixel in image:
*/
IG_DIB_pixel_get ( hIGear, 0, 0, &cPixelValue[0] );
```

Remarks:

If the pixel is 1-bit or 4-bit, the remaining bits of the byte will be set to zeroes.

If the pixel is 24-bit, 3 bytes are returned. These will be in the order Blue-Green-Red (unless you have changed the order of the image bitmap bytes such as by calling function IG_IP_swap_red_blue()).

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

1.3.1.2.7.27 IG_DIB_pixel_set

This function sets the pixel, at the location pointed to by lpPixel, into the image bitmap of image hIGear at coordinates (nXpos, nYpos).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pixel_set (
        HIGEAR hIGear,
        AT_PIXPOS nXpos,
        AT_PIXPOS nYpos,
        const LPAT_PIXEL lpPixel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of an image.
nXpos	AT_PIXPOS	X offset (in pixels) from beginning of raster line. First pixel on line is pixel number 0.
nYpos	AT_PIXPOS	Raster line number. 0 is top line.
pPixel	const LPAT_PIXEL	Far pointer to byte containing pixel, or to a 3-byte area if a 24-bit pixel.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;/* HIGEAR handle of image                                    */
AT_PIXEL cPixelValue[3];/* 3 bytes in case 24-bit image
*/
/* Set upper-leftmost pixel in image, to max pixel value:
*/
cPixelValue[0] = cPixelValue[1] = cPixelValue[2] = 255;
IG_DIB_pixel_set ( hIGear, 0, 0, &cPixelValue[0] );
```

Remarks:

The pixel is assumed to have the same number of Bits Per Pixel as the image, and if it is 1 or 4 bits, it is assumed to be right justified (that is, in the least significant bits) at location lpPixel. If the image is 24-bit, 3 bytes are transferred. Normally, these bytes will be in the order of Blue-Green-Red (unless the order of the image bitmap bytes has been changed by a call such as IG_IP_swap_red_blue()).

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

If the image you are modifying is 1-bit, you must convert the image from run-end encoded to a standard DIB, before you can set pixel values. Please see the section Accessing Image Pixels for details.

1.3.1.2.7.28 IG_DIB_pix_get

This function gets a pixel from the specified location in the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pix_get(
        HIGEAR hIGear,
        AT_PIXPOS xpos,
        AT_PIXPOS ypos,
        HIGPIXEL* lphPixel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image from which to get pixel.
xpos	AT_PIXPOS	X coordinate (0 to width-1).
ypos	AT_PIXPOS	Y coordinate (0 to height-1).
lphPixel	HIGPIXEL*	Returns object containing pixel data.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
    for (x = 0; x < w / 2; x++)
    {
        nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
        for (c = 0; c < nChannels; c++)
        {
            IG_image_channel_depth_get(hImage, c, &nDepth);
            nDepth = (1 << nDepth) - 1;
            inverted = nDepth - IG_pixel_value_get(hPix, c);
            IG_pixel_value_set(hPix, c, inverted);
        }
        nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
        IG_pixel_delete(hPix);
    }
```

```
nErrcount = IG_save_file(hImage, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

Remarks:

The pixel data is contained by a pixel object with handle of type HIGPIXEL. This pixel object stores values for each channel in the image.

1.3.1.2.7.29 IG_DIB_pix_set

This function sets a pixel at the specified location in the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_pix_set(
        HIGEAR hIGear,
        AT_PIXPOS xpos,
        AT_PIXPOS ypos,
        const HIGPIXEL hPixel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image into which to set pixel.
xpos	AT_PIXPOS	X coordinate (0 to width-1).
ypos	AT_PIXPOS	Y coordinate (0 to height-1).
hPixel	const HIGPIXEL	Pixel data to write.

Remarks:

The pixel data is contained by a pixel object with handle of type HIGPIXEL. This pixel object stores values for each channel in the image.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
    for (x = 0; x < w / 2; x++)
    {
        nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
        for (c = 0; c < nChannels; c++)
        {
            IG_image_channel_depth_get(hImage, c, &nDepth);
            nDepth = (1 << nDepth) - 1;
            inverted = nDepth - IG_pixel_value_get(hPix, c);
            IG_pixel_value_set(hPix, c, inverted);
        }
        nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
        IG_pixel_delete(hPix);
    }
nErrcount = IG_save_file(hImage, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

1.3.1.2.7.30 IG_DIB_raster_get

This function obtains an entire horizontal raster line of pixels from the DIB image bitmap.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_raster_get (
        HIGEAR hIGear,
        AT_PIXPOS nYpos,
        LPAT_PIXEL lpPixel,
        AT_MODE nFormat
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nYpos	AT_PIXPOS	Raster line number (0 is top line).
lpPixel	LPAT_PIXEL	Far pointer to first byte of area to receive the raster row of pixel values.
nFormat	AT_MODE	A variable of type AT_MODE (see accucnst.h) that tells whether the data being read in packed, unpacked, or RLE-compressed.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;                        /* HIGEAR handle of image
*/
AT_PIXEL                 cPixelValue[64];                       /* Needed to hold 500 1-
bit pixels                               */
AT_PIXPOS                nRaster;                      /* Index for the loop below
*/
AT_ERRCOUNT              nErrcount;                    /* Hold the returned error count
*/
AT_MODE          nFormat;
/* Obtain the top raster of a 1-bit image that's 500 pixels wide:               */
/* The pixels occupy 500/8 = 62.5 bytes.                                                */
nErrcount = IG_DIB_raster_get ( hIGear, 0, &cPixelValue[0], IG_PIXEL_PACKED);
/* Make the next 9 rows identical to the top row: */
for ( nRaster = 1; nRaster < 10; nRaster++ )
        nErrcount = IG_DIB_raster_set ( hIGear, nRaster, &cPixelValue[0], IG_PIXEL_PACKED
);
```

Remarks:

You may first make a call to IG_DIB_raster_size_get() in order to determine the size of buffer that you will need to hold the raster data. The format in which the data is returned, nFormat, tells ImageGear whether the data is packed, unpacked, or RLE-compressed. The values that nFormat may be set to are: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, and IG_PIXEL_RLE. IG_PIXEL_PACKED gives the storage format of a standard uncompressed DIB, which includes padding to a multiple of 4 bytes length. (If 1-bit or 4-bit, the pixels are packed 8 or 2 to a byte respectively, stored most-significant-bit-first.) 24-bit pixels are returned 3 bytes each, ordered Blue-Green-Red, with the row padded to a multiple of 4 bytes length.

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

1.3.1.2.7.31 IG_DIB_raster_set

This function sets a horizontal raster line of pixels into the DIB image bitmap.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_raster_set (
        HIGEAR hIGear,
        AT_PIXPOS nYpos,
        const LPAT_PIXEL lpPixel,
        AT_MODE nFormat
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of an image.
nYpos	AT_PIXPOS	Raster line number (0 is top line).
lpPixel	const LPAT_PIXEL	Far pointer to first byte of the pixel data to set.
nFormat	AT_MODE	A variable of type AT_MODE which indicates in which manner the raster data should be stored: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, or IG_PIXEL_RLE.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

See the example for function IG_DIB_raster_get().

Remarks:

nFormat should be set to the same format in which the rest of the data in the DIB is stored. If you choose IG_PIXEL_PACKED, your array of bytes should be padded with zeroes on the right, just as it is in the standard DIB format, such that its length is a multiple of 4.

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

If the image you are modifying is 1-bit, you must convert the image from run-end encoded to a standard DIB, before you can set pixel values. Please see the section Accessing Image Pixels for details.

1.3.1.2.7.32 IG_DIB_raster_size_get

This function calculates and returns the number of bytes required to hold a line (raster) from the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_raster_size_get(
        HIGEAR hIGear,
        AT_MODE nFormat,
        LPAT_DIMENSION lpSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image.
nFormat	AT_MODE	Format in which the raster data is stored: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, IG_PIXEL_RLE.
lpSize	LPAT_DIMENSION	A far pointer to a variable in which the size of the raster line (in bytes) is returned.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

You can use this function to determine the size of a block of memory to allocate before using IG_DIB_raster_get(), to avoid data overflow. The returned size will include allocation for buffering at the end of the rasters. (See the section Device-Independent Bitmaps (DIB) for more information on buffering in DIBs.) The nFormat variable determines in what form you would like to store the pixels in. The size will vary according to the storage method.

1.3.1.2.7.33 IG_DIB_resolution_get

This function copies resolution from DIB to lpResolution.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_resolution_get(
        HIGDIBINFO hDIB,
        AT_RESOLUTION* lpResolution
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.
lpResolution	AT_RESOLUTION*	Pointer to struct to which to copy DIB resolution.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_RESOLUTION res;      /* Returned image resolution */
nErrcount = IG_DIB_resolution_get(hDIB, &res);
```

1.3.1.2.7.34 IG_DIB_resolution_set

This function copies resolution from lpResolution to DIB.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_resolution_set(
        HIGDIBINFO hDIB,
        const AT_RESOLUTION* lpResolution
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.
lpResolution	const AT_RESOLUTION*	Pointer to struct to copy to DIB.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIB;        /* DIB info handle */
AT_RESOLUTION res;      /* Image resolution to be set */
/* Set resolution to 300 DPI */
res.units = IG_RESOLUTION_INCHES;
res.xResNumerator = res.yResNumerator = 300;
res.xResDenominator = res.yResDenominator = 1;
nErrcount = IG_DIB_resolution_set(hDIB, &res);
```

1.3.1.2.7.35 IG_DIB_resolution_units_get

This function has been deprecated and will be removed from the public API in a future release. Please use IG_DIB_resolution_get or IG_image_resolution_get instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_resolution_units_get(
        LPAT_MODE lpnResUnits
);
```

Arguments:

Name	Type	Description
lpnResUnits	LPAT_MODE	Pointer to AT_MODE where to return current identifier of resolution units.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;           /* HIGEAR handle of image */
AT_MODE nResUnits;
. . .IG_DIB_resolution_units_get( &nResUnits );. . .
```

Remarks:

ImageGear allows you to store the DIB resolution - biXPelsPerMeter and biYPelsPerMeter fields of AT_DIB structure - in different units depending on a global parameter; this function allows you to get the current value of this parameter. Valid values are IG_RESOLUTION_INCHES and IG_RESOLUTION_METERS. The default value is IG_RESOLUTION_METERS.

1.3.1.2.7.36 IG_DIB_resolution_units_set

This function has been deprecated and will be removed from the public API in a future release. Please use IG_DIB_resolution_set or IG_image_resolution_set instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_resolution_units_set(
        AT_MODE nDIBResUnits
);
```

Arguments:

Name	Type	Description
nDIBResUnits	AT_MODE	Value of type AT_MODE. Can be either IG_RESOLUTION_INCHES or IG_RESOLUTION_METERS.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_MODE nDIBResUnits
nDIBResUnits = IG_RESOLUTION_INCHES
...IG_DIB_resolution_units_set( nDIBResUnits );. . .
```

Remarks:

ImageGear allows you to store the DIB resolution (biXPelsPerMeter and biYPelsPerMeter fields of AT_DIB structure) in different units depending on a global parameter; this function allows you to set the current value of this parameter.

Valid values are IG_RESOLUTION_INCHES and IG_RESOLUTION_METERS. The default value is IG_RESOLUTION_METERS.

1.3.1.2.7.37 IG_DIB_row_get

This function obtains a consecutive row of nLength pixels beginning at coordinates (nXpos, xYpos) in the image bitmap.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_row_get (
        HIGEAR hIGear,
        AT_PIXPOS nXpos,
        AT_PIXPOS nYpos,
        AT_DIMENSION nLength,
        LPVOID lpPixel,
        AT_MODE nFormat
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nXpos	AT_PIXPOS	X offset (in pixels) from beginning of the raster line. First pixel on line is pixel number 0. Specify as -1 to obtain the entire raster line of pixels.
nYpos	AT_PIXPOS	Raster line number (0 is top line).
nLength	AT_DIMENSION	Number of consecutive pixels to obtain (entire raster line if nXpos = -1).
lpPixel	LPVOID	Far pointer to first byte of your area, at which the pixels obtained are to be stored.
nFormat	AT_MODE	Format in which the raster data is read: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, IG_PIXEL_RLE.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;                /* HIGEAR handle of image
*/
AT_PIXEL                 cPixelValue[300];                      /* To hold 100 pixels, in
case 24-bit*/
AT_PIXPOS                nRow;          /* Index for the loop below
*/
AT_DIMENSION             nRowLen;                /* How much of row to copy
*/
AT_ERRCOUNT              nErrcount;              /* Holds the returned error count
*/
/* Obtain leftmost 100 pixels of top raster of image:
*/
nRowLen = 100;
nErrcount = IG_DIB_row_get ( hIGear, 0, 0, nRowLen, &cPixelValue[0], IG_PIXEL_UNPACKED );
/* Make leftmost 100 pixels of next 9 rows identical:                   */
for ( nRow = 1; nRow < 10; nRow++ )
        nErrcount = IG_DIB_row_set (hIGear, 0, nRow,
                nRowLen, cPixelValue, IG_PIXEL_UNPACKED);
```

Remarks:

If nFormat is set to IG_PIXEL_UNPACKED, and the image is 1 or 4-bit, the pixels obtained are stored right justified, one to a byte, beginning at your byte pointed to by lpPixel. The unused bits of the bytes are set to zero. If the pixels are 24-bit, 3 bytes per pixel are returned, ordered Blue-Green-Red. A total of nLength pixels is transferred. (See the section Device-Independent Bitmaps (DIB)Understanding Bitmap Images for more details on pixel storage in DIBs.)

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

1.3.1.2.7.38 IG_DIB_row_set

This function writes a consecutive row of nLength pixels that begin at lpPixel, into image hIGear`s DIB image bitmap.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_DIB_row_set (
        HIGEAR hIGear,
        AT_PIXPOS nXpos,
        AT_PIXPOS nYpos,
        AT_DIMENSION nLength,
        const LPVOID lpPixel,
        AT_MODE nFormat
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nXpos	AT_PIXPOS	X offset (in pixels) from beginning of raster line. First pixel on line is pixel number 0.
nYpos	AT_PIXPOS	Raster line number. 0 is top line.
nLength	AT_DIMENSION	Number of consecutive pixels to transfer.
lpPixel	const LPVOID	Far pointer to byte at which the pixels to transfer begin.
nFormat	AT_MODE	Format in which the raster data is stored: IG_PIXEL_PACKED, IG_PIXEL_UNPACKED, IG_PIXEL_RLE.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

See the example for function IG_DIB_row_get().

Remarks:

The row is written into raster line nYpos, beginning at pixel offset nXpos. If nFormat is set to IG_PIXEL_UNPACKED, and the image is 1-bit or 4-bit, your pixels to be transferred should be one to a byte, right justified (that is, in the least significant bits). If a 24-bit image, each pixel should occupy 3 bytes, ordered Blue-Green-Red. (See the section Device-Independent Bitmaps (DIB) for more details on pixel storage in DIBs.)

(If (nXpos + nLength) is greater than the width of the image as indicated in the DIB header, an error will result.

ImageGear's pixel access functions consider the coordinates (0,0) to refer to the upper left-hand corner of the bitmap data. They do not follow the DIB's orientation, which considers (0,0) to refer to the lower left-hand corner of the bitmap.

If the image you are modifying is 1-bit, you will probably need to convert the image from run-end encoded to a standard DIB, before you can set pixel values. Please see the section Accessing Image Pixels for details.

1.3.1.2.7.39 IG_DIB_width_get

This function returns the width of the image.

Declaration:

```
AT_DIMENSION ACCUAPI IG_DIB_width_get(
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info handle.

Return Value:

Width of the image.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGDIBINFO hDIBInfo;    /* DIB info handle */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_DIMENSION width;     /* Returned height of image */
nErrcount = IG_image_DIB_info_get(hImage, &hDIBInfo);
width = IG_DIB_width_get(hDIBInfo);
```

1.3.1.2.8 Display Functions

This section provides information about the Display group of functions.

- IG_display_animation_delay_get
- IG_display_animation_delay_set
- IG_display_option_get
- IG_display_option_set
- IG_display_transparent_get
- IG_display_transparent_set
- IG_dspl_antialias_get
- IG_dspl_antialias_get_ex
- IG_dspl_antialias_set
- IG_dspl_antialias_set_ex
- IG_dspl_background_get
- IG_dspl_background_set
- IG_dspl_DDB_create
- IG_dspl_DDB_draw
- IG_dspl_DDB_import
- IG_dspl_device_to_image
- IG_dspl_device_to_image_d
- IG_dspl_dithering_get
- IG_dspl_dithering_set
- IG_dspl_document_print
- IG_dspl_document_print_custom
- IG_dspl_foreground_get
- IG_dspl_foreground_set
- IG_dspl_free_grp_id_get
- IG_dspl_gamma_correction_LUT_build
- IG_dspl_gamma_correction_set
- IG_dspl_grayscale_LUT_copy_get
- IG_dspl_grayscale_LUT_exists
- IG_dspl_grayscale_LUT_update_from
- IG_dspl_grp_reset
- IG_dspl_image_calc
- IG_dspl_image_draw
- IG_dspl_image_print
- IG_dspl_image_to_device
- IG_dspl_image_to_device_d
- IG_dspl_image_wipe
- IG_dspl_layout_get
- IG_dspl_layout_set
- IG_dspl_LUT_get
- IG_dspl_LUT_set
- IG_dspl_mapmode_get
- IG_dspl_mapmode_set
- IG_dspl_orientation_get
- IG_dspl_orientation_set
- IG_dspl_page_print
- IG_dspl_palette_create
- IG_dspl_palette_handle
- IG_dspl_palette_get
- IG_dspl_palette_set
- IG_dspl_PPM_correct_get
- IG_dspl_PPM_correct_set
- IG_dspl_resize_handle

- IG_dspl_ROP_get
- IG_dspl_ROP_set
- IG_dspl_scroll_get
- IG_dspl_scroll_handle
- IG_dspl_scroll_set
- IG_dspl_scroll_to
- IG_dspl_scroll_to_ex
- IG_dspl_transparency_get
- IG_dspl_transparency_set
- IG_dspl_zoom_get
- IG_dspl_zoom_set
- IG_dspl_zoom_to_rect

1.3.1.2.8.1 IG_display_animation_delay_get

This function obtains the animation delay setting for image hIGear, as set by IG_display_animation_delay_set().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_display_animation_delay_get (
        HIGEAR hIGear,
        LPUINT lpDelay
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpDelay	LPUINT	Far pointer to a UINT variable to receive current animation delay setting for this image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR            hIGear;                /* HIGEAR handle of image
*/
UINT             nDelay;                /* Will hold returned Delay setting
*/
IG_display_animation_delay_get ( hIGear, &nDelay );
```

Remarks:

ImageGear does not provide functions for creating animation. This function is for getting this value from the HIGEAR. You will need to write your own code to display the images in succession (animate).

1.3.1.2.8.2 IG_display_animation_delay_set

This function sets the animation delay for image hIGear (currently, this quantity is meaningful only with GIF format files).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_display_animation_delay_set (
        HIGEAR hIGear,
        UINT nDelay
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nDelay	UINT	Animation delay, in milliseconds. A setting of 100 would mean 10 frames per second.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR            hIGear;                        /* HIGEAR handle of image
*/
/* Set for 5 frames per second: */
IG_display_animation_delay_set ( hIGear, 200 );
```

Remarks:

ImageGear does not provide functions for creating animation. This function is for setting this value in the HIGEAR. You will need to write your own code to display the images in succession (animate).

1.3.1.2.8.3 IG_display_option_get

This function allows you to get the current display option settings for either a specific HIGEAR image or to get the settings that will be inherited by each new HIGEAR image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_display_option_get (
        HIGEAR hIGear,
        AT_MODE nOption,
        LPVOID lpOption,
        LPVOID lpReserved
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image, or NULL.
nOption	AT_MODE	A integer of type AT_MODE that tells ImageGear what type of display option for which to return a value. The options are defined in accucnst.h and their names begin with IG_DISPLAY_OPTION_.
lpOption	LPVOID	A long pointer to VOID data returned as display option data.
lpReserved	LPVOID	This argument is reserved for future use. Please set to NULL for now.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The value that you set for hIGear determines whether the option settings will be retrieved from a specific image or a global setting for all new HIGEAR images. If you set hIGear to a specific image, the settings for that image will be returned. If you set hIGear to NULL, global settings for all new HIGEAR images will be returned.

See the description under IG_display_option_set() for more details about the display settings.

1.3.1.2.8.4 IG_display_option_set

This function allows you to set the current display option settings for either a specific HIGEAR image or to set the settings that will be inherited by each new HIGEAR image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_display_option_set (
        HIGEAR  hIGear,
        AT_MODE nOption,
        const LPVOID lpOption,
        LPVOID lpReserved
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image or NULL.
nOption	AT_MODE	A integer of type AT_MODE that tells ImageGear which display option to set. The options are defined in accucnst.h and their names begin with IG_DISPLAY_OPTION_.
lpOption	const LPVOID	A long pointer to VOID display option data. See details below.
lpReserved	LPVOID	This argument is reserved for future use. Please set to NULL for now.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR            hIGear;                        /* HIGEAR handle of image */
IG_display_option_set( hIGear, IG_DISPLAY_OPTION_DOWNSHIFT, (LPVOID)0, NULL);
```

Remarks:

You may set these options for a specific HIGEAR or for all newly created HIGEAR images. To set the options for a particular HIGEAR image, set hIGear to the appropriate handle. To set the options for all newly created HIGEAR images, set hIGear to NULL.

nOption must be set to one of the IG_DISPLAY_OPTION_ constants defined in accucnst.h. lpOption should be set to the value of the option. This will vary depending on what option you are setting. See the list below.

Currently, this function is useful for 16-bit grayscale images only. However, constants will be added to support all bit depths. See accucnst.h for new constants beginning with IG_DISPLAY_OPTION_.

Currently available options include:

- IG_DISPLAY_OPTION_DOWNSHIFT: This option is for 16-bit grayscale DIBs only. It has no effect on any other image bit depth. The value passed in to nOption must be in the range of 0 to 16 (the value is cast to a LPVOID - the address of this value is not passed in). This value specifies how far each 16-bit pixel should be downshifted before the least significant word is taken for display. All remaining bits in the high word are discarded. Setting this option turns off the IG_DISPLAY_OPTION_LUT option.
- IG_DISPLAY_OPTION_LUT: This option is for 16-bit grayscale DIBs only. It has no effect on any other image bit depth. The value passed in to nOption must be a pointer to a 16x8 LUT (64K of memory). This table should be filled with values that allow ImageGear to display 16-bit grayscale pixels on a 8-bit display. Setting this option

turns off the IG_DISPLAY_OPTION_DOWNSHIFT option.

Additional options include:

- IG_DISPLAY_OPTION_OFFSCREEN_DRAW: If this parameter is TRUE then the display code optimizes the drawing when the ART component is used to prevent flashing. This a bit slower, but the visual quality is better. Otherwise, each redraw operation is directly displayed, with flashing possible.
- IG_DISPLAY_OPTION_DDB_OPTIMIZE: If this parameter is TRUE then monochrome DDB is created from 1bpp HIGEAR image. Otherwise, a compatible bitmap to the current display is created.
- IG_DISPLAY_OPTION_OFFSCREEN_WIDTH/IG_DISPLAY_OPTION_OFFSCREEN_HEIGHT: Default values (0, 0) means that GetDeviceCaps(hDC,HORZRES) / GetDeviceCaps(hDC,VERTRES) will be used, respectively. These options specify the size of offscreen drawing surface. Set it to the size of a single monitor. In case of a dual monitor system you need to set IG_DISPLAY_OPTION_OFFSCREEN_WIDTH / IG_DISPLAY_OPTION_OFFSCREEN_HEIGHT to the total size of your dual monitor screen.

1.3.1.2.8.5 IG_display_transparent_get

Use this function to get image transparent color.

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_display_transparent_get(
        HIGEAR hIGear,
        LPAT_RGB lpRGB,
        LPAT_BOOL lpEnabled
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRGB	LPAT_RGB	Returns the RGB color values through which transparency is currently set.
lpEnabled	LPAT_BOOL	Returns current condition of transparency: TRUE - transparency enabled; FALSE - transparency disabled.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.8.6 IG_display_transparent_set

Use this function to set image transparent color.

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_display_transparent_set(
        HIGEAR hIGear,
        LPAT_RGB lpRGB,
        LPAT_BOOL lpEnabled
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRGB	LPAT_RGB	Provide the RGB color values through which transparency have to be set.
lpEnabled	LPAT_BOOL	Enables or disables image transparency: TRUE - transparency enabled; FALSE - transparency disabled.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

You have to use this API to write out a file with a transparency color set. For example when save image transparency information to a GIF file format.

1.3.1.2.8.7 IG_dspl_antialias_get

This function returns the current anti-alias settings.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_antialias_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPAT_MODE lpnAliasFlags,
        [OUT] LPUINT lpnThreshold
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group from which to get option.
lpnAliasFlags	LPAT_MODE	Pointer to where AliasMode is to be received. If NULL, then this parameter is ignored.
lpnThreshold	LPUINT	Pointer to where ThresholdValue is to be received. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;     /* HIGEAR handle of image  */
DWORD           nGrpID;     /* display group identifier */
AT_MODE nAliasFlags; /* alias flags  */
UINT            nThreshold;  /* alias threshold */
 ...
IG_dspl_antialias_get( hIGear, nGrpID, &nAliasFlags, &nThreshold );
 ...
```

Remarks:

Possible values are listed in the description of the IG_dspl_antialias_set() function.

1.3.1.2.8.8 IG_dspl_antialias_get_ex

This function returns the current anti-alias settings.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_antialias_get_ex(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPAT_MODE lpnAliasFlags,
        [OUT] LPUINT lpnThreshold
        [OUT] LPUINT lpnQuality
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group from which to get option.
lpnAliasFlags	LPAT_MODE	Pointer to where AliasMode is to be received. If NULL, then this parameter is ignored.
lpnThreshold	LPUINT	Pointer to where ThresholdValue is to be received. If NULL, then this parameter is ignored.
lpnQuality	LPUINT	Pointer to where color anti-aliasing quality value is to be received. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;     /* HIGEAR handle of image  */
DWORD           nGrpID;     /* display group identifier */
AT_MODE nAliasFlags; /* alias flags  */
UINT            nThreshold;  /* alias threshold */
UINT nQuality /*alias quality */
 ...
IG_dspl_antialias_get_ex( hIGear, nGrpID, &nAliasFlags, &nThreshold, &nQuality);
 ...
```

Remarks:

Possible values are listed in the description of the IG_dspl_antialias_set_ex() function.

1.3.1.2.8.9 IG_dspl_antialias_set

This function sets new anti-alias settings.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_antialias_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nAliasFlags,
        [IN] INT nThreshold
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set option.
nAliasFlags	AT_MODE	New value of AliasMode to set. Possible value is one of the following: • IG_DSPL_ANTIALIAS_NONE • IG_DSPL_ANTIALIAS_SCALE_TO_GRAY • IG_DSPL_ANTIALIAS_PRESERVE_BLACK • IG_DSPL_ANTIALIAS_PRESERVE_WHITE with OR combination of one additional flag: • IG_DSPL_ANTIALIAS_SUBSAMPLE
nThreshold	INT	Specifies display aliasing threshold level (AliasThreshold). Please see Dithering, Anti-Aliasing, and Palette Handling.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;     /* HIGEAR handle of image  */
DWORD           nGrpID;     /* display group identifier */
 ...
/* sets scale to gray algorithm with subsampling  */
IG_dspl_antialias_set( hIGear, nGrpID, IG_DSPL_ANTIALIAS_SCALE_TO_GRAY|
IG_DSPL_ANTIALIAS_SUBSAMPLE, 50 );
 ...
```

1.3.1.2.8.10 IG_dspl_antialias_set_ex

This function sets new anti-alias settings.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_antialias_set_ex(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nAliasFlags,
        [IN] INT Threshold,
        [IN] INT nColorQuality
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set option.
nAliasFlags	AT_MODE	New value of AliasMode to set. It is an OR combination of following elements: ● Scale-Up interpolation (re-sampling) ● Scale-Down interpolation (anti-aliasing) For 1 bit images anti-aliasing element is a one of constants: ● IG_DSPL_ANTIALIAS_NONE ● IG_DSPL_ANTIALIAS_SCALE_TO_GRAY ● IG_DSPL_ANTIALIAS_PRESERVE_BLACK ● IG_DSPL_ANTIALIAS_PRESERVE_WHITE with OR combination of one additional flag IG_DSPL_ANTIALIAS_SUBSAMPLE Anti-aliasing for all non-1 bit color spaces is activated with the IG_DSPL_ANTIALIAS_COLOR flag. Scale-Up interpolation for all non-1 bit color spaces is specified by the IG_DSPL_ANTIALIAS_RESAMPLE_BILINE flag.
nThreshold	INT	It indicates the threshold value of the amount of selected color to include for IG_DSPL_ANTIALIAS_PRESERVE_XXXX scale-down interpolations. Please see Dithering, Anti-Aliasing, and Palette Handling.
nColorQuality	INT	Specifies quality of color scale-down interpolation (anti-aliasing). Valid values are from 0 to 100 and specify the percent of image pixels taken into account to produce one display pixel. Zero value means automatic quality choice according with image resolution and display settings.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;     /* HIGEAR handle of image  */
DWORD           nGrpID;     /* display group identifier */
 ...
/* sets scale to gray algorithm with subsampling  */
IG_dspl_antialias_set_ex( hIGear, nGrpID, IG_DSPL_ANTIALIAS_SCALE_TO_GRAY|
IG_DSPL_ANTIALIAS_SUBSAMPLE, 50, 80);
 ...
```

1.3.1.2.8.11 IG_dspl_background_get

This function returns the background settings' current values.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_background_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPAT_MODE lpnBkMode,
        [OUT] LPAT_RGB lpBkColor,
        [OUT] HBITMAP FAR* lphBrush
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of the group to use.
lpnBkMode	LPAT_MODE	Pointer to where BkMode options are returned. If NULL, then this parameter is ignored.
lpBkColor	LPAT_RGB	Pointer to where BkColor option is returned. If NULL, then this parameter is ignored.
lphBrush	HBITMAP FAR*	Pointer to where BkBrush option is returned. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;      /* HIGEAR handle of image  */
DWORD           nGrpID;      /* display group identifier */
AT_MODE nBkMode;    /* backgound mode  */
AT_RGB BkColor;    /* background color  */
HBITMAP hBrush;     /* background mask  */
 ...
IG_dspl_background_get( hIGear, nGrpID, &nBkMode, &BkColor, &hBrush );
 ...
```

Remarks:

Possible values are listed in the description of function IG_dspl_background_set().

1.3.1.2.8.12 IG_dspl_background_set

This function sets the background settings.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_background_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nBkMode,
        [IN] const LPAT_RGB lpBkColor,
        [IN] HBITMAP hBrush
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of the group in which to set options.
nBkMode	AT_MODE	IG_DSPL_BACKGROUND_NONE or a combination of the following flags: ● IG_DSPL_BACKGROUND_UNDER_IMAGE ● IG_DSPL_BACKGROUND_BEYOND_IMAGE
lpBkColor	const LPAT_RGB	New value of BkColor to set. If NULL, then this parameter is ignored.
hBrush	HBITMAP	New value of BkBrush option to set. Please note that the old bitmap is not deleted, and the application code is responsible for removing the old value of this option.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;    /* HIGEAR handle of image  */
DWORD           nGrpID;  /* display group identifier  */
 ...
/* disable background under image and beyond image  */
IG_dspl_background_set( hIGear, nGrpID, IG_DSPL_BACKGROUND_NONE, NULL, NULL );
 ...
```

1.3.1.2.8.13 IG_dspl_DDB_create

This function creates a DDB of the image.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_DDB_create(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HDC hDC,
        [IN] AT_DIMENSION nWidth,
        [IN] AT_DIMENSION nHeight,
        [IN] BOOL bExport,
        [OUT] HBITMAP FAR* lphBitmap,
        [OUT] HPALETTE FAR* lphPalette
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which the options are stored.
hDC	HDC	Handle of device context with which the DDB should be compatible. If NULL, then the DDB will be compatible with the desktop's device context.
nWidth	AT_DIMENSION	Width of DDB that is to be created.
nHeight	AT_DIMENSION	Height of DDB that is to be created.
bExport	BOOL	Boolean parameter which specifies whether to delete the source image or not. If TRUE then the hIGear image will be deleted after the DDB is created, but if FALSE then the image is left unchanged.
lphBitmap	HBITMAP FAR*	Pointer to where to return the DDB. For Mac OS X, DDB is CGImageRef object.
lphPalette	HPALETTE FAR*	Pointer to where to return the palette handle for this DDB. For Mac OS X, this parameter should be NULL.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
DWORD nGrpID; /* display group identifier */
HBITMAP hBitmap; /* handle of bitmap */
- (IBAction)mnuFileConvertToDDB:(id)sender {
    if(IG_image_is_valid(hIGear))
    {
        if(hBitmap != 0)
        {
            CGImageRelease(hBitmap);
            hBitmap = 0;
        }
        AT_DIMENSION width, height;
        IG_image_dimensions_get( hIGear, &width, &height, NULL );
        IG_dspl_DDB_create( hIGear, 0, NULL, width, height, TRUE, &hBitmap, NULL);
        hIGear = 0;
        // Update main view
        [mainScrollViewOutlet setNeedsDisplay:YES];
    }
}
```

Remarks:

This function always uses all the display options specified by dwGrpIDgroup and assumes that the output device has a 32bpp RGB color format, but the client area of the output device is a rectangle of nWidth x nHeight size.

1.3.1.2.8.14 IG_dspl_DDB_draw

This function displays the specified bitmap at the specified coordinates in the specified device context.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_DDB_draw(
       HWND hWnd,
       HDC hDC,
       HBITMAP hBitmap,
       HPALETTE hPalette,
       AT_PIXPOS x,
       AT_PIXPOS y
);
```

Arguments:

Name	Type	Description
hWnd	HWND	This reference is unused.
hDC	HDC	HDC to which to display the specified hBitmap. For Mac OS X, HDC is CGContextRef object.
hBitmap	HBITMAP	HBITMAP to display on the specified hDC. For Mac OS X, HBITMAP is CGImageRef object.
hPalette	HPALETTE	HPALETTE of the specified hBitmap. For Mac OS X, this parameter should be NULL.
x	AT_PIXPOS	The x coordinate of the destination hDC to display the hBitmap.
Y	AT_PIXPOS	The y coordinate of the destination hDC to display the hBitmap.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

To determine the number of errors currently on the error stack use IG_error_check. After fetching all error information you need using IG_error_get, use IG_error_clear to clear the stack.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.8.15 IG_dspl_DDB_import

This function imports a HBITMAP into a HIGEAR instance.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_DDB_import(
        HDC hDC,
        HBITMAP hBitmap,
        HPALETTE hPalette,
        LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
hDC	HDC	HDC to use when processing the hBitmap. For Mac OS X, it is not used.
hBitmap	HBITMAP	HBITMAP to import to the HIGEAR instance. For Mac OS X, HBITMAP is CGImageRef object.
hPalette	HPALETTE	HPALETTE of the specified hBitmap. For Mac OS X, this parameter should be NULL.
lphIGear	LPHIGEAR	HIGEAR instance to which the hBitmap will be imported.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

To determine the number of errors currently on the error stack use IG_error_check. After fetching all error information you need using IG_error_get, use IG_error_clear to clear the stack.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.8.16 IG_dspl_device_to_image

This function translates an array of points from device coordinates to image coordinates.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_device_to_image(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] HDC hDC,
        [IN/OUT] LPAT_POINT lpPoint,
        [IN] UINT nCount
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of the group from which to get the display options.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
hDC	HDC	Handle of the device context used for drawing. This can be NULL, but if a calculation is necessary for a printer device context, then you should provide the real value.
lpPoint	LPAT_POINT	Pointer to an array of points that should be translated.
nCount	UINT	Number of elements in the lpPoint array.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
    NSView* nsView = self;
    HIGEAR hIGear; /* HIGEAR handle of image */
    DWORD nGrpID = 0; /* display group identifier */
    AT_POINT p[2]; /* array of point to translate */
    NSRect rc = [nsView frame];

    /* calculate coordinates in image coordinate space of current client rectangle of the
     window */
    p[0].x = rc.origin.x;;
    p[0].y = rc.origin.y;
    p[1].x = rc.origin.x + rc.size.width - 1;
    p[1].y = rc.origin.y + rc.size.height - 1;

    IG_dspl_device_to_image( hIGear, nGrpID, (__bridge HWND)nsView, NULL, p, 2 );
```

Remarks:

This function takes into account all display parameters including orientation and current scrolling position.

1.3.1.2.8.17 IG_dspl_device_to_image_d

This function translates an array of points in DOUBLE float-points format from the device coordinates to the image coordinates.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_device_to_image_d(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] HDC hDC,
        [IN/OUT] LPAT_DPOINT lpPoint,
        [IN] UINT nCount
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of the group from which to get display options.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
hDC	HDC	Handle of the device context used for drawing. This can be NULL, but if a calculation is necessary for printer device context, then you should provide the real value.
lpPoint	LPAT_DPOINT	Pointer to an array of points in DOUBLE float-point format that should be translated.
nCount	UINT	Number of elements in the lpPoint array.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

This function takes into account all display parameters, including the orientation and current scrolling position.

See also the function IG_dspl_image_to_device().

1.3.1.2.8.18 IG_dspl_dithering_get

This function returns the current dithering flags.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_dithering_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPAT_MODE lpnDitherFlags
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group from which to get option.
lpnDitherFlags	LPAT_MODE	Pointer where DitherMode is to be received.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;    /* HIGEAR handle of image  */
DWORD           nGrpID;    /* display group identifier */
AT_MODE nDitherFlags; /* dither flags */
 ...
IG_dspl_dithering_get( hIGear, nGrpID, &nDitherFlags );
 ...
```

Remarks:

All possible values are listed in the description of function IG_dspl_dithering_set().

1.3.1.2.8.19 IG_dspl_dithering_set

This function sets new dithering options.

Declaration:

```
AT_ERRCOUNT  ACCUAPI   IG_dspl_dithering_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nDitherFlags
);
```

Arguments:

hIGear	ImageGear handle of image.
dwGrpID	Identifier of group from which to set dithering.
nDitherFlags	New value of DitherMode to set. Possible values include: ● IG_DSPL_DITHER_AUTO ● IG_DSPL_DITHER_TO_8BPP ● IG_DSPL_DITHER_TO_4BPP ● IG_DSPL_DITHER_TO_1BPP ● IG_DSPL_DITHER_NONE with OR combination from two flags: ● IG_DSPL_DITHER_FIXED_PALETTE ● IG_DSPL_DITHER_NETSCAPE_PALETTE

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR   hIGear;     /* HIGEAR handle of image */
DWORD   nGrpID;     /* display group identifier  */
 ...
/* if device is palette based then dither to fixed palette */
IG_dspl_dithering_set( hIGear, nGrpID, IG_DSPL_DITHER_AUTO| IG_DSPL_DITHER_FIXED_PALETTE
);
 ...
```

1.3.1.2.8.20 IG_dspl_document_print

This function allows you to print an array of images and specify the number of images per width (row) and per height (column) of the page.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_document_print(
        [IN] const LPHIGEAR lphIGear,
        [IN] UINT nImageCount,
        [IN] DWORD dwGrpID,
        [IN] HDC hDC,
        [IN] UINT nImagesPerWidth,
        [IN] UINT nImagesPerHeight,
        [IN] DOUBLE dblXSpace,
        [IN] DOUBLE dblYSpace,
        [IN] BOOL bDirectToDriver,
        [IN] LPFNIG_IMAGESPOOLED lpfnImageSpooled,
        [IN] LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
lphIGear	const LPHIGEAR	Array of ImageGear image handles to print.
nImageCount	UINT	Number of elements in lphIGear array.
dwGrpID	DWORD	Identifier of group from which to get options for printing for each image.
hDC	HDC	Handle of printer device context on which to draw images.
nImagesPerWidth	UINT	Number of images that should be placed in row.
nImagesPerHeight	UINT	Number of images that should be placed in column.
dblXSpace	DOUBLE	Horizontal destination between images in row in page's width relative coordinates. This means the actual destination in device coordinates is calculated as: xSpace = PageWidth*dblXSpace.
dblYSpace	DOUBLE	Vertical destination between images in column in page's height relative coordinates. This means the actual destination in device coordinates is calculated as: ySpace = PageHeight*dblYSpace.
bDirectToDriver	BOOL	If TRUE, then ImageGear does not perform image scaling, but uses the operating system's and driver's capabilities for this. If FALSE then ImageGear performs the scaling.
lpfnImageSpooled	LPFNIG_IMAGESPOOLED	Callback function that will be called after each image is printed.
lpPrivateData	LPVOID	Private data that will be passed to lpfnImageSpooled callback function in first parameter.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR lphIGear[10]; /* array of HIGEAR handles of images  */
DWORD nGrpID;       /* display group identifier  */
```

```
BOOL bDirect;  /* direct to driver flag  */
PRINTDLG pd;      /* print dialog structure  */
INT nPrivateInfo;
 ...
case ID_FILE_PRINT:
         ...
        if( PrintDlg(&pd) )
        {
 ...
IG_dspl_document_print( lphIGear, 10, nGrpID, pd.hDC, 2, 3, 0.05, 0.05, bDirect,
ImageSpooled, &nPrivateInfo );
 ...
        }
         ...
        break;
 ...
BOOL ACCUAPI  ImageSpooled(
        LPVOID   lpPrivate,  /* Private data passed in */
        UINT      nImageNumber, /* Current image being spooled (1 based) */
        UINT     nPageNumber   /* Current page number being spooled    */
        )
{
         ...
        return TRUE;   /* return false to cancel printing */
}
 ...
```

Remarks:

lpfnImageSpooled function will be called after each image is printed and can use the lpPrivateDataparameter as private data storage. bDirectToDriver parameter allows you to perform image scaling inside of ImageGear or leave this task to the printer driver and operating system. Usually, direct to driver printing (bDirectToDriver=TRUE) results in smaller output size and it works faster but not using it produces better quality and allows you to use such ImageGear capabilities as anti-aliasing during printing.

Special predefined option group IG_GRP_DEFAULT_PRINT can be used to print an image with the most common parameters.

1.3.1.2.8.21 IG_dspl_document_print_custom

This function allows you to print an array of images and customize each image's layout on the page.

Declaration:

```
AT_ERRCOUNT    ACCUAPI    IG_dspl_document_print_custom(
        [IN] const LPHIGEAR lphIGear,
        [IN] UINT nImageCount,
        [IN] DWORD dwGrpID,
        [IN] HDC hDC,
        [IN] UINT nImagesPerPage,
        [IN] const LPAT_DRECTANGLE lpImagesLayout,
        [IN] BOOL bDirectToDriver,
        [IN] LPFNIG_IMAGESPOOLED lpfnImageSpooled,
        [IN] LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
lphIGear	const LPHIGEAR	Array of ImageGear image handles to print.
nImageCount	UINT	Number of elements in lphIGear array.
dwGrpID	DWORD	Identifier of group from which to get the options for printing for each image.
hDC	HDC	Handle of printer device context on which to draw the images.
nImagesPerPage	UINT	Number of images that should be placed in a single page.
lpImagesLayout	const LPAT_DRECTANGLE	Array of length nImagesPerPage of rectangles where an i-th rectangle specifies how the i-th image of this page is located on that page. Each rectangle is calculated in page-relative units, and as the actual page resolutions are obtained, it translates the rectangles into real coordinates and assigns values to ClipRect according to the following rules: ClipRect.x = lpLayout[i].x*nPageWidth ClipRect.y = lpLayout[i].y*nPageHeight ClipRect.width = lpLayout[i].width*nPageWidth ClipRect.height = lpLayout[i].height*nPageHeight
bDirectToDriver	BOOL	If TRUE, then ImageGear does not perform image scaling but uses the operating system's and driver's capabilities for this. If FALSE then ImageGear performs the scaling.
lpfnImageSpooled	LPFNIG_IMAGESPOOLED	Callback function that will be called after each image is printed.
lpPrivateData	LPVOID	Private data that will be passed to the lpfnImageSpooled callback function in the first parameter.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           lphIGear[10];  /* array of HIGEAR handles of images */
DWORD           nGrpID;        /* display group identifier  */
BOOL            bDirect;        /* direct to driver flag  */
AT_DRECTANGLE
```

```
                Layout[2];   /* array describes image layout on single page */
PRINTDLG                pd;         /* print dialog structure   */
INT             nPrivateInfo;
 ...
case ID_FILE_PRINT:
         ...
        if( PrintDlg(&pd) )
        {
 ...
/* place one page at the left top of the page              */
Layout[0].x = 0.01; Layout[0].y = 0.01;
Layout[0].width = 0.48; Layout[0].height = 0.48;
/* and second page at the right bottom                     */
Layout[1].x = 0.51; Layout[1].y = 0.51;
Layout[1].width = 0.48; Layout[1].height = 0.48;
IG_dspl_document_print_custom( lphIGear, 10, nGrpID, pd.hDC, 2, Layout, bDirect,
ImageSpooled, &nPrivateInfo );
        }        ...
        break; ...
BOOL    ACCUAPI  ImageSpooled(
        LPVOID   lpPrivate,      /* Private data passed in   */
        UINT     nImageNumber, /* Current image being spooled (1 based) */
        UINT     nPageNumber  /* Current page number being spooled  */
        )
{        ...
        return  TRUE;   /* return FALSE to cancel printing  */
}
```

Remarks:

lpfnImageSpooled function will be called after each image is printed and you can use the lpPrivateData parameter as private data storage. bDirectToDriver parameter allows you to perform image scaling inside of ImageGear or leave this task to the printer driver and operating system. Usually, direct to driver printing (bDirectToDriver=TRUE) results in smaller output size and it works faster, but not using it produces better quality and allows you to use such ImageGear capabilities as anti-aliasing during printing.

Special predefined option group IG_GRP_DEFAULT_PRINT can be used to print an image with the most common parameters.

1.3.1.2.8.22 IG_dspl_foreground_get

This function returns the foreground color.

Declaration:

```
AT_ERRCOUNT  ACCUAPI   IG_dspl_foreground_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPAT_RGB lpFrColor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	The ImageGear handle of an image.
dwGrpID	DWORD	Identifier of the group to use.
lpFrColor	LPAT_RGB	Pointer to where the FrColor option is returned. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;         /* HIGEAR handle of image   */
DWORD nGrpID;         /* display group identifier  */
AT_RGB FrColor;        /* foreground color  */
 ...
IG_dspl_foreground_get( hIGear, nGrpID, &FrColor );
 ...
```

1.3.1.2.8.23 IG_dspl_foreground_set

This function sets the foreground color.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_foreground_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] const LPAT_RGB lpFrColor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of an image.
dwGrpID	DWORD	Identifier of the group in which to set options.
lpFrColor	const LPAT_RGB	The new value of FrColor to set. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;     /* HIGEAR handle of image   */
DWORD           nGrpID;      /* display group identifier  */
AT_RGB          FrColor;
 ...
FrColor.r = FrColor.g = FrColor.b;
IG_dspl_foreground_set( hIGear, nGrpID, &FrColor );
...
```

1.3.1.2.8.24 IG_dspl_free_grp_id_get

This function searches and returns the first free unused group identifier.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_free_grp_id_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwMin,
        [OUT] LPDWORD lpdwFreeGrpId
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwMin	DWORD	Minimum value from which ImageGear should start to search.
lpdwFreeGrpId	LPDWORD	Pointer to where to return free group identifier.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image   */
DWORD nGrpID;   /* display group identifier  */
 ...
IG_dspl_free_grp_id_get( hIGear, 0, &nGrpID );
 ...
```

Remarks:

This function also sets a flag so the returned group is marked as used and the next call to this function with the same parameters returns another group id. To mark a group as unused and reset it to the default values, call function IG_dspl_grp_reset().

1.3.1.2.8.25 IG_dspl_gamma_correction_LUT_build

This function builds look-up tables from given contrast, brightness and gamma values.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_gamma_correction_LUT_build(
        [IN] DOUBLE dblContrast,
        [IN] DOUBLE dblBrightness,
        [IN] DOUBLE dblGamma,
        [OUT] LPBYTE lpLUT
);
```

Arguments:

Name	Type	Description
dblContrast	DOUBLE	Contrast value to use in calculations. You can use any value.
dblBrightness	DOUBLE	Brightness value to use in calculations. Must be within the -255.0 to +255.0 range.
dblGamma	DOUBLE	Gamma value to use in calculations. Must be greater than 0.0, however the most useful values are in the range from 1.8 to 2.2.
lpLUT	LPBYTE	Pointer to 256 byte array in which to calculate the look-up table.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
BYTE lut[256];    /* lookup array  */
 ...
IG_dspl_gamma_correction_LUT_build( 2.0, 120.0, 2.0, lut );
```

1.3.1.2.8.26 IG_dspl_gamma_correction_set

This function takes contrast (dblContrast), brightness (dblBrightness) and gamma (dblGamma) parameters and calculates look-up tables accordingly and sets them into the corresponding options.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_gamma_correction_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nFlags,
        [IN] DOUBLE dblContrast,
        [IN] DOUBLE dblBrightness,
        [IN] DOUBLE dblGamma
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set look-up tables.
nFlags	AT_MODE	Specify which look-up tables to set. Possible values are 0 or a combination of these flags: • IG_DSPL_R_CHANNEL - if this flag is set then the RedLut option is to be set. • IG_DSPL_G_CHANNEL - if this flag is set then the GreenLut option is to be set. • IG_DSPL_B_CHANNEL - if this flag is set then the BlueLut option is to be set. • The constant IG_DSPL_ALL_CHANNELS is defined for convenience and can be used to set all three options. `#define IG_DSPL_ALL_CHANNELS (IG_DSPL_R_CHANNEL|IG_DSPL_G_CHANNEL|IG_DSPL_B_CHANNEL)`
dblContrast	DOUBLE	Contrast value to use in calculations. You can use any value.
dblBrightness	DOUBLE	Brightness value to use in calculations. Must be in the range from -255.0 to +255.0.
dblGamma	DOUBLE	Gamma value to use in calculations. Must be greater than 0.0; however the most useful values are in the range from 1.8 to 2.2.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;    /* HIGEAR handle of image  */
DWORD           nGrpID;    /* display group identifier */
 ...
IG_dspl_gamma_correction_set( hIGear, nGrpID, IG_DSPL_ALL_CHANNELS, 2.0, 120.0, 2.0 );
 ...
```

Remarks:

All look-up tables specified in the nFlagsparameter are initialized with the same value based on the dblContrast, dblBrightness, and dblGamma values.

1.3.1.2.8.27 IG_dspl_grayscale_LUT_copy_get

This function returns a copy of display grayscale LUT, if it exists in the specified display settings group.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_grayscale_LUT_copy_get(
        HIGEAR hIGear,
        DWORD dwGrpID,
        HIGLUT* lpLUT
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle.
dwGrpID	DWORD	Display group ID.
lpLUT	HIGLUT*	New LUT handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

1.3.1.2.8.28 IG_dspl_grayscale_LUT_exists

This function checks whether the display settings group has a grayscale LUT attached.

Declaration:

```
AT_BOOL ACCUAPI IG_dspl_grayscale_LUT_exists(
        HIGEAR hIGear,
        DWORD dwGrpID
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle.
dwGrpID	DWORD	Display group ID.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

1.3.1.2.8.29 IG_dspl_grayscale_LUT_update_from

This function updates (creates if not present) a grayscale LUT for the image display.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_grayscale_LUT_update_from(
        HIGEAR hIGear,
        DWORD dwGrpID,
        HIGLUT lut
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle.
dwGrpID	DWORD	Display group ID.
lut	HIGLUT	LUT handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

1.3.1.2.8.30 IG_dspl_grp_reset

This function resets all the options of the specified group to its default values.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_grp_reset(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group to reset.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
    HIGEAR hIGear; /* HIGEAR handle of image */ DWORD nGrpID; /* display group identifier */
...
    IG_dspl_grp_reset( hIGear, nGrpID ); ...
```

Remarks:

Please note the application code is responsible for freeing memory allocated for options such as TranspMaskandBkBrush. This function does not free it.

This function also removes the flag that marks this group as "used." It may be set again by any display function that specifies using this group identifier.

1.3.1.2.8.31 IG_dspl_image_calc

This function calculates DisplayedImageRect, which is the exact rectangle in which the image is to be scaled.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_image_calc(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] HDC hDC,
        [OUT] LPAT_RECTANGLE lpActualDeviceRect
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle from which to get options and pixel data.
dwGrpID	DWORD	Identifier of group from which to get display options.
hWnd	HWND	Handle of window where image is to be displayed. Can be NULL, but in this case some display options may be calculated incorrectly if they are not set explicitly.
hDC	HDC	Handle of the device context where image is to be drawn.
lpActualDeviceRect	LPAT_RECTANGLE	Pointer to the rectangle in which to copy DisplayedImageRect. This rectangle is calculated no matter whether hDC=NULL or not. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR                   hIGear;         /* HIGEAR handle of image
*/
DWORD                   nGrpID;         /* Display group identifier
*/
AT_RECTANGLE                     DisplayedRect;
...
case WM_PAINT:
        BeginPaint(hWnd, &ps);
        if (IG_image_is_valid(hIGear))
                IG_dspl_image_calc( hIGear, nGrpID, hWnd, ps.hdc, &DisplayedRect );
        EndPaint(hWnd, &ps);
        break;
```

Remarks:

All options for drawing the image are taken from the dwGrpID group. This function does not perform any drawing, whether hDC is NULL or not.

1.3.1.2.8.32 IG_dspl_image_draw

This function draws an image onto a destination device context.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_image_draw(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] HDC hDC,
        [OUT] LPAT_RECTANGLE lpActualDeviceRect
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle from which to get options and pixel data.
dwGrpID	DWORD	Identifier of group from which to get display options.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is to be displayed. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
hDC	HDC	Handle of the device context on which to draw the image. Can be NULL, and in this case no actual drawing is performed but all parameters are calculated and updated. This may be useful in calculating DisplayedImageRect in the next parameter of this function.
lpActualDeviceRect	LPAT_RECTANGLE	Pointer to the rectangle in which to copy DisplayedImageRect. This rectangle is calculated no matter whether hDC=NULL or not. If the value is NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image
*/
DWORD nGrpID = 0; /* Display group identifier
*/
...
- (void)drawRect:(NSRect)dirtyRect
{
    if(IG_image_is_valid(hIGear))
    {
        CGContextRef myContext = [[NSGraphicsContext currentContext] graphicsPort];
        if([NSGraphicsContext currentContextDrawingToScreen])
            IG_dspl_image_draw(hIGear, 0, (__bridge HWND)self, (HDC)myContext, NULL);
        else
            IG_dspl_image_draw(hIGear, 0, NULL, (HDC)myContext, NULL);
    }
}
```

Remarks:

All options regarding how to draw the image are taken from the dwGrpID group. This function can also be used for the actual device rectangle calculation. If hDC=NULL then all parameters are calculated and updated but no actual drawing is performed.

1.3.1.2.8.33 IG_dspl_image_print

This function draws an image onto the printer device context.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_image_print(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HDC hDC,
        [IN] BOOL bDirectToDriver
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
DwGrpID	DWORD	Identifier of group from which to get image options.
hDC	HDC	Handle of device context on which to draw the image.
bDirectToDriver	BOOL	If TRUE then ImageGear does not perform image scaling but use the operating system's and driver's capabilities for this. If FALSE then ImageGear performs the scaling.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image
*/
DWORD nGrpID; /* display group identifier
*/
BOOL bDirect ; /* direct to driver flag
*/
- (void)drawRect:(NSRect)dirtyRect
{

    if(IG_image_is_valid(hIGear))
    {
        // Get device context
        CGContextRef myContext = [[NSGraphicsContext currentContext] graphicsPort];
        if([NSGraphicsContext currentContextDrawingToScreen])
            // Draw the image to the screen
            IG_dspl_image_draw(hIGear, 0, (__bridge HWND)self, (HDC)myContext, NULL);
        else
        {
            // Set printing resolution
            AT_INT printRes = 200;
            IG_gctrl_item_set( "PRINT.RESOLUTION", AM_TID_INT, &printRes, sizeof(AT_INT),
NULL );
            // Print the image
            IG_dspl_image_print(hIGear, 0, (HDC)myContext, bDirect);
        }
    }
}
```

Remarks:

Print resolution is controlled with “PRINT.RESOLUTION” Global Control Parameter. bDirectToDriver parameter allows you to either perform image scaling inside of ImageGear or leave this task to the printer driver and operating system. Usually, direct to driver printing (bDirectToDriver=TRUE) results in smaller output size and works faster, but not using it produces better quality and allows you to use ImageGear capabilities such as anti-aliasing during printing.

Special predefined option group IG_GRP_DEFAULT_PRINT can be used to print an image with the most common parameters.

1.3.1.2.8.34 IG_dspl_image_to_device

This function translates an array of pointers from image coordinates into device coordinates.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_image_to_device(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] HDC hDC,
        [IN/OUT] LPAT_POINT lpPoint,
        [IN] UINT nCount
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group from which to get display options.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
hDC	HDC	Handle of the device context used for drawing. Can be NULL, but if it is necessary to perform a calculation for the printer device context, then a real value should be provided.
lpPoint	LPAT_POINT	Pointer to an array of points that should be translated.
nCount	UINT	Number of elements in lpPoint array.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
NSView* nsView = self;

HIGEAR hIGear; /* HIGEAR handle of image
*/
DWORD nGrpID = 0; /* display group identifier
*/
AT_POINT p[2]; /* array of point to translate
*/
AT_RECTANGLE ImageRect; /* image rectangle
*/
...
/* calculates device coordinates of current image rectangle */
IG_dspl_layout_get( hIGear, nGrpID, &ImageRect, NULL, NULL, NULL, NULL, NULL, NULL );
p[0].x = ImageRect.x;
p[0].y = ImageRect.y;
p[1].x = ImageRect.x + ImageRect.width - 1;
p[1].y = ImageRect.y + ImageRect.height - 1;
IG_dspl_image_to_device( hIGear, nGrpID, (__bridge HWND)nsView, NULL, p, 2 );
...
```

Remarks:

This function takes into account all display parameters including orientation and current scrolling position.

1.3.1.2.8.35 IG_dspl_image_to_device_d

This function translates an array of points in DOUBLE float-point format from the image coordinates into device coordinates.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_image_to_device_d(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] HDC hDC,
        [IN/OUT] LPAT_DPOINT lpPoint,
        [IN] UINT nCount
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of an image.
dwGrpID	DWORD	Identifier of the group from which to get display options.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
hDC	HDC	Handle of the device context used for drawing. Can be NULL, but if it is necessary to perform a calculation for the printer device context, then a real value should be provided.
lpPoint	LPAT_DPOINT	Pointer to an array of points in DOUBLE float-point format that should be translated.
nCount	UINT	Number of elements in the lpPoint array.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

This function takes into account all display parameters, including orientation and current scrolling position.

See also the function IG_dspl_device_to_image().

1.3.1.2.8.36 IG_dspl_image_wipe

This function changes the image in window hWnd from the hIGearBefore image to the hIGearAfter image.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_image_wipe(
        [IN] HIGEAR hIGearBefore,
        [IN] DWORD dwGrpBefore,
        [IN] HIGEAR hIGearAfter,
        [IN] DWORD dwGrpAfter,
        [IN] HWND hWnd,
        [IN] AT_MODE nWipeStyle,
        [IN] LONG nGranularity,
        [IN] LONG lTime
);
```

Arguments:

Name	Type	Description
hIGearBefore	HIGEAR	HIGEAR handle of an image in the window before the wipe.
dwGrpBefore	DWORD	Display group identifier used for hIGearBefore.
hIGearAfter	HIGEAR	HIGEAR handle of an image to be in window after the wipe.
dwGrpAfter	DWORD	Display group identifier used for hIGearAfter.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
nWipeStyle	AT_MODE	Transition style, such as IG_WIPE_LEFTTORIGHT. See below. (Check updates to accucnst.h for new wipe styles.)
nGranularity	LONG	Size of each square region, in pixels. A typical value is 5.
lTime	LONG	The time, in milliseconds, between wipes.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGearBefore; /* Handle of image already being displayed */
HIGEAR hIGearAfter; /* Handle of image to replace it */
NSView* nsView; /* Pointer to NSView object */
AT_ERRCOUNT nErrcount; /* Returned count of errors */
/* Perform a "sparkle" wipe, changing 16 x 16 pixel areas in each step:*/
nErrcount = IG_dspl_image_wipe ( hIGearBefore, IG_GRP_DEFAULT, hIGearAfter,
IG_GRP_DEFAULT,  (__bridge HWND)nsView, IG_WIPE_SPARKLE, 16, 5 );
/* See sample application WIPES for a complete example */
```

Remarks:

The transition is accomplished according to the nWipeStyle style. Before you call this function, the first image should already be in the window. This function will set the wipe style, and then generate a WM_PAINT message. The IG_dspl_image_draw() call while processing this WM_PAINT message will perform the transition to the new image. nWipeStyle Constants include:

IG_WIPE_LEFTTORIGHT	Left-to-Right wipe.
IG_WIPE_RIGHTTOLEFT	Right-to-Left wipe.
IG_WIPE_UP_TO_DOWN	Up-to-Down wipe.
IG_WIPE_DOWN_TO_UP	Down-to-Up wipe.
IG_WIPE_SPARKLE	Sparkle Transition.
IG_WIPE_ULTOLRDIAG	Upper Left to Lower Right wipe.
IG_WIPE_LRTOULDIAG	Lower Right to Upper Left wipe.
IG_WIPE_URTOLLDIAG	Upper Right to Lower Left wipe.
IG_WIPE_LLTOURDIAG	Lower Left to Upper Right wipe.
IG_WIPE_CLOCK	Clockwise wipe.
IG_WIPE_SPARKLE_CLOCK	Clockwise wipe with sparkles.
IG_WIPE_DOUBLE_CLOCK	Two simultaneous clockwise wipes, 180 degrees apart.
IG_WIPE_SLIDE_RIGHT	New image slides in from the left.
IG_WIPE_SLIDE_LEFT	New image slides in from the right.
IG_WIPE_SLIDE_UP	New image slides in from the bottom.
IG_WIPE_SLIDE_DOWN	New image slides in from the top.
IG_WIPE_RANDOM_BARS_DOWN	Vertical bars of old image fall to reveal new image.
IG_WIPE_RAIN	Vertical lines of new image cover over old, like paint running down the side of a bucket.
IG_WIPE_BOOK	Book wipe.
IG_WIPE_ROLL	Old image rolls in from right to left.
IG_WIPE_UNROLL	New image rolls out from left to right.
IG_WIPE_EXPAND_PROPORTIONAL	New image expands from the center of old image in diagonal directions.
IG_WIPE_EXPAND_HORIZONTAL	New image expands from the center of old image in horizontal directions.
IG_WIPE_EXPAND_VERTICAL	New image expands from the center of old image in vertical directions.
IG_WIPE_STRIPS_HORIZONTAL	New image appears as expanding horizontal strips.
IG_WIPE_STRIPS_VERTICAL	New image appears as expanding vertical strips.
IG_WIPE_CELLS	New image appears as expanding square cells.
IG_WIPE_BALL	New image appears as tracks of spirally moving balls.
IG_WIPE_GEARS	New image appears as tracks of moving ImageGear's icons.

1.3.1.2.8.37 IG_dspl_layout_get

This function returns the current values of layout parameters.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_layout_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPAT_RECTANGLE lpImageRect,
        [OUT] LPAT_RECTANGLE lpDeviceRect,
        [OUT] LPAT_RECTANGLE lpClipRect,
        [OUT] LPAT_MODE lpnFitMode,
        [OUT] LPAT_MODE lpnAlignMode,
        [OUT] LPAT_MODE lpnAspectMode,
        [OUT] LPDOUBLE lpdAspectValue
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image handle from which to search the option group.
dwGrpID	DWORD	Identifier of group which to get layout options.
lpImageRect	LPAT_RECTANGLE	Pointer to the rectangle in which to copy the value of ImageRect option. If NULL then it is ignored. If ImageRect is not set then an empty rectangle is returned.
lpDeviceRect	LPAT_RECTANGLE	Pointer to the rectangle in which to copy DeviceRect. If NULL, then the parameter is ignored. If an empty rectangle is returned then it is not set and it will be calculated every time from the destination device.
lpClipRect	LPAT_RECTANGLE	Pointer to the rectangle in which to copy ClipRect. If NULL, then the parameter is ignored. If an empty rectangle returned then it is not set and it will be calculated every time from the destination device.
lpnFitMode	LPAT_MODE	Assigns the current FitMode. If NULL then the parameter is ignored.
lpnAlignMode	LPAT_MODE	Assigns the current AlignMode. If NULL then the parameter is ignored.
lpnAspectMode	LPAT_MODE	Assigns the current AspectMode. If NULL then the parameter is ignored.
lpdAspectValue	LPDOUBLE	Assigns the current AspectValue. If NULL then the parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;     /* HIGEAR handle of image  */
DWORD           nGrpID;      /* display group identifier */
AT_RECTANGLE             ImageRect;    /* Image rectangle  */
AT_RECTANGLE            DeviceRect;    /* Device rectangle  */
AT_RECTANGLE             ClipRect;      /* Clip rectangle  */
AT_MODE     nFitMode;                /* Fit mode  */
AT_MODE     nAlignMode;             /* Align mode  */
AT_MODE     nAspectMode;            /* Aspect mode  */
DOUBLE      dAspectValue;    /* Aspect value  */
```

```
 ...
/* get all layout parameters */
IG_dspl_layout_get( hIGear, nGrpID, &ImageRect, &DeviceRect, &ClipRect, &nFitMode,
&nAlignMode, &nAspectMode, &dAspectValue );
 ...
/* get only device rectangle and fit mode */
IG_dspl_layout_get( hIGear, nGrpID, NULL, &DeviceRect, NULL, &nFitMode, NULL, NULL, NULL
);
 ...
```

Remarks:

If a parameter is a rectangle then all empty rectangle is returned if it is not set. For a list of possible values see function IG_dspl_layout_set().

1.3.1.2.8.38 IG_dspl_layout_set

This function sets layout parameters.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_layout_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] DWORD nFlags,
        [IN] const LPAT_RECTANGLE lpImageRect,
        [IN] const LPAT_RECTANGLE lpDeviceRect,
        [IN] const LPAT_RECTANGLE lpClipRect,
        [IN] AT_MODE nFitMode,
        [IN] AT_MODE nAlignMode,
        [IN] AT_MODE nAspectMode,
        [IN] DOUBLE dblAspectValue
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set layout options.
nFlags	DWORD	Combination of flags where each one specifies if the appropriate parameter is to be used or ignored: IG_DSPL_IMAGE_RECT - if (nFlags&IG_DSPL_IMAGE_RECT)!=0 then the lpImageRect parameter of this function is not ignored IG_DSPL_DEVICE_RECT - if (nFlags&IG_DSPL_DEVICE_RECT)!=0 then the lpDeviceRect parameter of this function is not ignored IG_DSPL_CLIP_RECT - if (nFlags&IG_DSPL_CLIP_RECT)!=0 then the lpClipRect parameter of this function is not ignored IG_DSPL_FIT_MODE - if (nFlags&IG_DSPL_FIT_MODE)!=0 then the nFitMode parameter of this function is not ignored IG_DSPL_ALIGN_MODE - if (nFlags&IG_DSPL_ALIGN_MODE)!=0 then the nAlignMode parameter of this function is not ignored IG_DSPL_ASPECT_MODE - if (nFlags&IG_DSPL_ASPECT_MODE)!=0 then the nAspectMode parameter of this function is not ignored IG_DSPL_ASPECT_VALUE - if (nFlags&IG_DSPL_ASPECT_VALUE)!=0 then the dblAspectValue parameter of this function is not ignored
lpImageRect	const LPAT_RECTANGLE	A new value of ImageRect option to set. If NULL then it is reset with the default value which means the whole image will be used.
lpDeviceRect	const LPAT_RECTANGLE	A new value of DeviceRect option to set. If NULL, then it is reset with the default value (an empty rectangle). This means it is calculated every time from the destination output device.
lpClipRect	const LPAT_RECTANGLE	A new value of ClipRect option to set. If NULL, then it is reset with the default value (an empty rectangle). This means it is calculated every time and assigned to the whole client area of the destination output device.
nFitMode	AT_MODE	New value of FitMode option to set. Possible values are: • IG_DSPL_FIT_TO_DEVICE • IG_DSPL_FIT_TO_WIDTH • IG_DSPL_FIT_TO_HEIGHT • IG_DSPL_ACTUAL_SIZE
nAlignMode	AT_MODE	New value of AlignMode option to set. Possible value is OR combination of one constant that sets the horizontal alignment: • IG_DSPL_ALIGN_X_LEFT • IG_DSPL_ALIGN_X_CENTER • IG_DSPL_ALIGN_X_RIGHT

		and one that sets the vertical alignment: ● IG_DSPL_ALIGN_Y_TOP ● IG_DSPL_ALIGN_Y_CENTER ● IG_DSPL_ALIGN_Y_BOTTOM
nAspectMode	AT_MODE	New value of AspectMode option to set. Possible values are: ● IG_DSPL_ASPECT_FIXED ● IG_DSPL_ASPECT_NOT_FIXED
dblAspectValue	DOUBLE	Sets new value to AspectValue option. Possible value may be any positive double.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;    /* HIGEAR handle of image  */
DWORD           nGrpID;     /* display group identifier */
AT_RECTANGLE             ImageRect;  /* Image rectangle   */
AT_RECTANGLE             DeviceRect;  /* Device rectangle   */
AT_RECTANGLE             ClipRect;     /* Clip rectangle  */
AT_MODE          nFitMode;     /* Fit mode     */
AT_MODE          nAlignMode;    /* Align mode     */
AT_MODE          nAspectMode;   /* Aspect mode     */
DOUBLE          dAspectValue;      /* Aspect value  */
AT_MODE          nFlags;        /* flags which specify what to set */
/* set all layout parameters */
nFlags =
IG_DSPL_IMAGE_RECT|IG_DSPL_DEVICE_RECT|IG_DSPL_CLIP_RECT|IG_DSPL_FIT_MODE|IG_DSPL_ALIGN_M
ODE|IG_DSPL_ASPECT_MODE|IG_DSPL_ASPECT_VALUE;
IG_dspl_layout_get( hIGear, nGrpID, &ImageRect, &DeviceRect, &ClipRect, nFitMode,
nAlignMode, nAspectMode, dAspectValue );
 ... ...
/* reset image, device and clip rectangles to its default values */
nFlags = IG_DSPL_IMAGE_RECT|IG_DSPL_DEVICE_RECT|IG_DSPL_CLIP_RECT;
IG_dspl_layout_set( hIGear, nGrpID, nFlags, NULL, NULL, NULL, 0, 0, 0, 0.0 );
 ...
/* set new fit mode */
IG_dspl_layout_set( hIGear, nGrpID, IG_DSPL_FIT_MODE, NULL, NULL, NULL,
IG_DSPL_FIT_TO_WIDTH, 0, 0, 0.0 );
 ...
/* set align mode so that the image located at right bottom of device rectangle */
IG_dspl_layout_set( hIGear, nGrpID, IG_DSPL_ALIGN_MODE, NULL, NULL, NULL, 0,
IG_DSPL_ALIGN_X_RIGHT| IG_DSPL_ALIGN_Y_BOTTOM, 0, 0.0 );
 ...
```

Remarks:

If some of the values are out of range, then it does not change, and an error is returned.

1.3.1.2.8.39 IG_dspl_LUT_get

This function returns the current red, green and blue look-up tables lpRLUT, lpGLUT, or lpBLUT (should be either NULL or valid pointers to a 256 byte array).

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_LUT_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPBYTE lpRLUT,
        [OUT] LPBYTE lpGLUT,
        [OUT] LPBYTE lpBLUT
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group from which to get options.
lpRLUT	LPBYTE	Pointer to where the current red look-up table (RedLut value) is to be copied. If NULL, then this parameter is ignored.
lpGLUT	LPBYTE	Pointer where the current green look-up table (GreenLut value) is to be copied. If NULL, then this parameter is ignored.
lpBLUT	LPBYTE	Pointer where the current blue look-up table (BlueLut value) is to be copied. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR    hIGear;          /* HIGEAR handle of image  */
DWORD    nGrpID;         /* display group identifier  */
BYTE     r_lut[256];       /* red lookup array     */
BYTE     g_lut[256];     /* green lookup array     */
 ...
IG_dspl_LUT_get( hIGear, nGrpID, r_lut, g_lut, NULL );
 . .
```

1.3.1.2.8.40 IG_dspl_LUT_set

This function assigns new values to red, green, and blue look-up tables.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_LUT_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nFlags,
        [IN] const LPBYTE lpRLUT,
        [IN] const LPBYTE lpGLUT,
        [IN] const LPBYTE lpBLUT
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set LUT options.
nFlags	AT_MODE	Specify which look-up tables to set. Possible value is 0 or a combination of flags IG_DSPL_R_CHANNEL - if this flag is set then the lpRLUT parameter of this function is not ignored IG_DSPL_G_CHANNEL - if this flag is set then the lpGLUT parameter of this function is not ignored IG_DSPL_B_CHANNEL - if this flag is set then the lpBLUT parameter of this function is not ignored The constant IG_DSPL_ALL_CHANNELS is defined for convenience and can be used to set all three channels. `#define IG_DSPL_ALL_CHANNELS (IG_DSPL_R_CHANNEL|IG_DSPL_G_CHANNEL|IG_DSPL_B_CHANNEL)`
lpRLUT	const LPBYTE	Pointer to a 256 element array of a red look-up table to set. If NULL, then identity the array is assigned to RedLut option.
lpGLUT	const LPBYTE	Pointer to a 256 element array of a green look-up table to set. If NULL, then identity the array is assigned to GreenLut option.
lpBLUT	const LPBYTE	Pointer to a 256 element array of a blue look-up table to set. If NULL, then identity the array is assigned to BlueLut option.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR    hIGear;         /* HIGEAR handle of image  */
DWORD    nGrpID;        /* display group identifier  */
BYTE     lut[256];      /* lookup array  */
INT      i;
 ...
/* set inverted look-up table */
for( i = 0; i<256; i++ )
        lut[i] = 255 - i;
IG_dspl_LUT_set( hIGear, nGrpID, IG_DSPL_ALL_CHANNELS, lut, lut, lut );
```

Remarks:

ImageGear always makes a copy of lpRLUT, lpGLUT and lpBLUT in case they are needed, but does not assign pointers directly so that the application should cleanup the memory allocated for them.

1.3.1.2.8.41 IG_dspl_mapmode_get

This function returns the current map mode and logical coordinate system where parameters such as ClipRect, DeviceRect and most others (except ImageRect) are stored.

Declaration:

```
AT_ERRCOUNT    ACCUAPI   IG_dspl_mapmode_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPDWORD lpdwMapMode,
        [OUT] LPAT_RECTANGLE lpViewport,
        [OUT] LPAT_RECTANGLE lpWindow
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of Image.
dwGrpID	DWORD	Identifier of group from which to get map mode options.
lpdwMapMode	LPDWORD	Pointer to the current value of option MapMode.
lpViewport	LPAT_RECTANGLE	Pointer to the current value of rectangle Viewport.
lpWindow	LPAT_RECTANGLE	Pointer to the current value of rectangle Window.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;     /* HIGEAR handle of image  */
DWORD            nGrpID;     /* display group identifier */
DWORD            dwMapMode;    /* map mode   */
AT_RECTANGLE            Viewport;     /* view port values    */
AT_RECTANGLE             Window;       /* window values  */
 ...
IG_dspl_mapmode_get( hIGear, nGrpID, &dwMapMode, &Viewport, &Window );
SetMapMode( dwMapMode );
SetWindowOrgEx( hDC, Window.x, Window.y, NULL );
SetWindowExtEx( hDC, Window.width, Window.height, NULL );
SetViewportOrgEx( hDC, Viewport.x, Viewport.y, NULL );
SetViewportExtEx( hDC, Viewport.width, Viewport.height, NULL );
 ...
```

1.3.1.2.8.42 IG_dspl_mapmode_set

This function sets the current map mode and logical coordinate system.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_mapmode_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] DWORD dwMapMode,
        [IN] const LPAT_RECTANGLE lpViewport,
        [IN] const LPAT_RECTANGLE lpWindow
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set map mode options.
dwMapMode	DWORD	New value of option MapMode to assign.
LpViewport	const LPAT_RECTANGLE	New value of Viewport option to assign.
LpWindow	const LPAT_RECTANGLE	New value of Window option to assign.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;     /* HIGEAR handle of image  */
DWORD           nGrpID;      /* display group identifier  */
DWORD           dwMapMode;       /* map mode   */
AT_RECTANGLE            Viewport;         /* view port values  */
AT_RECTANGLE             Window;          /* window values  */
POINT           p;
SIZE            s;
 ...
/* get current mapping parameters */
dwMapMode = GetMapMode( hDC );
GetViewportOrgEx( hDC, &p );
Viewport.x = p.x;
Viewport.y = p.y;
GetViewportExtEx( hDC, &s );
Viewport.width = s.cx;
Viewport.height = s.cy;
GetWindowOrgEx( hDC, &p );
Window.x = p.x;
Window.y = p.y;
GetWindowExtEx( hDC, &s );
Window.width = s.cx;
Window.height = s.cy;
IG_dspl_mapmode_set( hIGear, nGrpID, dwMapMode, &Viewport, &Window );
 ...
```

Remarks:

ImageGear will perform all calculations with the assumption that the specified logical system is used for the device coordinates.

1.3.1.2.8.43 IG_dspl_orientation_get

This function returns the current orientation mode.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_orientation_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPAT_MODE lpnOrientMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group from which to retrieve the orientation mode.
lpnOrientMode	LPAT_MODE	Where to copy OrientMode value. If NULL then this parameter ignored.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR              hIGear;       /* HIGEAR handle of image  */
DWORD             nGrpID;        /* display group identifier  */
AT_MODE            nOrientMode;      /* Orientation mode  */
 ...
IG_dspl_orientation_get( hIGear, nGrpID, &nOrientMode );
 ...
```

Remarks:

Possible values are listed in the IG_dspl_orientation_set() function.

1.3.1.2.8.44 IG_dspl_orientation_set

This function sets the new orientation mode.

Declaration:

```
AT_ERRCOUNT  ACCUAPI IG_dspl_orientation_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nOrientMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image handle where to set orientation mode.
dwGrpID	DWORD	Identifier of the group in which to set the orientation.
nOrientMode	AT_MODE	New OrientMode value to set. Possible values are: • IG_DSPL_ORIENT_TOP_LEFT • IG_DSPL_ORIENT_LEFT_TOP • IG_DSPL_ORIENT_RIGHT_TOP • IG_DSPL_ORIENT_TOP_RIGHT • IG_DSPL_ORIENT_BOTTOM_RIGHT • IG_DSPL_ORIENT_RIGHT_BOTTOM • IG_DSPL_ORIENT_LEFT_BOTTOM • IG_DSPL_ORIENT_BOTTOM_LEFT

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR        hIGear;       /* HIGEAR handle of image   */
DWORD         nGrpID;       /* display group identifier  */
 ...
/* rotate image orientation by 90 degree   */
IG_dspl_orientation_set( hIGear, nGrpID, IG_DSPL_ORIENT_TOP_RIGHT );
 ...
```

1.3.1.2.8.45 IG_dspl_page_print

This function draws an image onto the printer device context within the specified rectangle.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_page_print(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HDC hDC,
        [IN] const LPAT_DRECTANGLE lpLayout,
        [IN] BOOL bDirectToDriver
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group from which to get printing options.
hDC	HDC	Handle of printer device context on which to draw the image.
lpLayout	const LPAT_DRECTANGLE	Rectangle which specifies how the image is located on the page. This rectangle is calculated in page-relative units, and as actual page resolutions are obtained it translates the rectangle into real coordinates and assigns ClipRect according to the following rules: ClipRect.x = lpLayout->x*nPageWidth ClipRect.y = lpLayout->y*nPageHeight ClipRect.width = lpLayout->width*nPageWidth ClipRect.height = lpLayout->height*nPageHeight
bDirectToDriver	BOOL	If TRUE then ImageGear does not perform image scaling but uses the operating system's and driver's capabilities for this. If FALSE then ImageGear performs the scaling.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;        /* HIGEAR handle of image  */
DWORD           nGrpID;        /* display group identifier   */
BOOL            bDirect;       /* direct to driver flag  */
AT_DRECTANGLE
                Layout;
PRINTDLG                 pd;          /* print dialog structure  */
 ...
case ID_FILE_PRINT:
         ...
        if( PrintDlg(&pd) )
        {
 ...
/* print image in the middle of the page and at 0.5 of width and height of the page */
Layout.x = 0.25;                        Layout.y = 0.25;
Layout.width = 0.5; Layout.height = 0.5;
IG_dspl_page_print( hIGear, nGrpID, pd.hDC, &Layout, bDirect );
...
        }
         ...
        break;
...
```

Remarks:

Printing resolution depends on the current printer setting. The bDirectToDriver parameter allows you to perform image scaling inside of ImageGear or leave this task to the printer driver and operating system. Usually, direct to driver printing (bDirectToDriver=TRUE) results in smaller output size and it works faster, but not using it produces better quality and allows you to use ImageGear capabilities such as anti-aliasing during printing.

Special predefined option group IG_GRP_DEFAULT_PRINT can be used to print an image with the most common parameters.

1.3.1.2.8.46 IG_dspl_palette_create

This function calculates all the display parameters, the way IG_dspl_image_draw() does, and produces the same palette that IG_dspl_image_draw realizes before drawing onto hDC; it returns DevicePalette if it is not set with a custom value.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_palette_create(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] HDC hDC,
        [OUT] HPALETTE FAR* lphPalette
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
DwGrpID	DWORD	Identifier of group containing the palette options to use.
HWnd	HWND	Handle of window where the image is drawn.
HDC	HDC	Handle of device context on which to draw.
LphPalette	HPALETTE FAR*	Pointer to where the palette handle should be returned.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;      /* HIGEAR handle of image  */
DWORD           nGrpID;      /* display group identifier */
HPALETTE                hPalette;     /* handle of palette  */
 ...
IG_dspl_palette_create( hIGear, nGrpID, hWnd, hDC, &hPalette );
 ...
```

1.3.1.2.8.47 IG_dspl_palette_handle

This function is designed to be used to handle palette messages from the operating system.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_palette_handle(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] HDC hDC,
        [IN] AT_MODE nPalMode,
        [OUT] LPBOOL lpbRealized
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group that should be used.
hWnd	HWND	Valid handle of window where image is drawn and from which palette message is to be handled.
hDC	HDC	Handle of device context where image is to be drawn. If NULL, then it will automatically be retrieved from hWnd.
nPalMode	AT_MODE	New value of PaletteMode to set. Possible value is either IG_DSPL_PALETTE_HIGH or IG_DSPL_PALETTE_LOW. Please note that this new value is assigned only if the current value of PaletteMode is not IG_DSPL_PALETTE_DISABLE. If the current value is IG_DSPL_PALETTE_DISABLE then the function does not perform any action (to change this, use IG_dspl_palette_set() function).
lpbRealized	LPBOOL	Pointer to the BOOL parameter where TRUE will be assigned if new palette has been realized and FALSE if otherwise. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;      /* HIGEAR handle of image   */
DWORD           nGrpID;      /* display group identifier */
 ...
case WM_QUERYNEWPALETTE:
        /* let Image Gear handle palette management */
IG_dspl_palette_handle(hIGear, nGrpID, hWnd, NULL, IG_DSPL_PALETTE_HIGH, &bRealized );
        return bRealized;
case WM_PALETTECHANGED:
        /* let Image Gear handle palette management */
IG_dspl_palette_handle(hIGear, nGrpID, hWnd, NULL, IG_DSPL_PALETTE_LOW, NULL );
        break;
 ...
```

h2>Remarks:
For the Windows platform, this function should be called when WM_QUERYNEWPALETTE and WM_PALETTECHANGED messages are processed (see the sample below). Handling of those messages is extremely important for palette

based devices.

1.3.1.2.8.48 IG_dspl_palette_get

This function returns the current palette settings.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_palette_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPAT_MODE lpnPalMode,
        [OUT] HPALETTE FAR* lphPalette
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group from which to get palette options.
lpnPalMode	LPAT_MODE	Pointer to where PaletteMode is to be received. If NULL, then this parameter is ignored.
lphPalette	HPALETTE FAR*	Pointer to where DevicePalette is to be received. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;      /* HIGEAR handle of image   */
DWORD            nGrpID;      /* display group identifier  */
AT_MODE          nPalMode;    /* palette mode  */
HPALETTE                  hPalette;    /* palette handle */
 ...
IG_dspl_palette_get( hIGear, nGrpID, &nPalMode, &hPalette );
 ...
```

1.3.1.2.8.49 IG_dspl_palette_set

This function sets the new palette options.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_palette_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nPalMode,
        [IN] HPALETTE hPalette
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set palette options.
nPalMode	AT_MODE	New value of PaletteMode to set. Possible values are: • IG_DSPL_PALETTE_HIGH • IG_DSPL_PALETTE_LOW • IG_DSPL_PALETTE_DISABLE
hPalette	HPALETTE	New value of DevicePalette option. Possible value is NULL or valid OS-dependent palette handle. Please note that the application code is responsible for removing all the resources allocated for DevicePalette if it is not NULL.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;    /* HIGEAR handle of image   */
DWORD            nGrpID;    /* display group identifier  */
 ...
/* disable palette realization */
IG_dspl_palette_set( hIGear, nGrpID, IG_DSPL_PALETTE_DISABLE, NULL );
 ...
```

1.3.1.2.8.50 IG_dspl_PPM_correct_get

This function returns the current value of options which specify either to use the image resolution or the image dimension to calculate the aspect ratio.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_PPM_correct_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPBOOL lpbEnable
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in from which to get options.
LpbEnable	LPBOOL	Pointer to where to get the current value of option PPMCorrect.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR             hIGear;        /* HIGEAR handle of image  */
DWORD            nGrpID;      /* display group identifier    */
BOOL             bPPMEnable;    /* PPM corect flag value  */
 ...
IG_dspl_PPM_correct_get ( hIGear, nGrpID, &bPPMEnable );
 ...
```

1.3.1.2.8.51 IG_dspl_PPM_correct_set

This function sets the new value of the PPMCorrect option.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_PPM_correct_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] BOOL bEnable
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group for which to set options.
bEnable	BOOL	New value for PPMCorrect option. If TRUE then the resolution will be used to calculate the aspect ratio, but if FALSE then the image dimension will be used.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR              hIGear;     /* HIGEAR handle of image    */
DWORD             nGrpID;     /* display group identifier */
 ...
IG_dspl_PPM_correct_set ( hIGear, nGrpID, TRUE );
 ...
```

1.3.1.2.8.52 IG_dspl_resize_handle

This function is to be called during WM_SIZE message handling and it recalculates all the parameters according to the new window width and height.

Declaration:

```
AT_ERRCOUNT  ACCUAPI IG_dspl_resize_handle(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group which is used to display the image on the hWnd.
hWnd	HWND	Handle of window where the image is displayed.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;       /* HIGEAR handle of image  */
DWORD           nGrpID;        /* display group identifier */
 ...
case WM_SIZE:
        /* handle window resizing */
        IG_dspl_resize_handle( hIGear, nGrpID, hWnd );
        break;
 ...
```

1.3.1.2.8.53 IG_dspl_ROP_get

This function gets the ROP (raster-operation) code used to display images on Windows platforms.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_ROP_get(
        HIGEAR hIGear,
        DWORD dwGrpID,
        LPDWORD lpdwROP
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
dwGrpID	DWORD	Identifier of display group from which to get ROP code.
lpdwROP	LPDWORD	Returned ROP code.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR       hIGear;      /* HIGEAR handle of image */
DWORD        nGrpID;      /* display group identifier */
DWORD        dwROP;       /* retrieved ROP code */
 ...
/* get ROP code */
IG_dspl_ROP_get( hIGear, nGrpID, &dwROP );
 ...
```

Remarks:

This code determines how the source image pixels are combined with the destination area pixels. The default ROP is SRCCOPY, which overwrites the destination area with the source image. Other codes are described in the documentation for the Windows GDI function BitBlt.

1.3.1.2.8.54 IG_dspl_ROP_set

This function sets the ROP (raster-operation) code to use when displaying images on Windows platforms.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_dspl_ROP_set(
        HIGEAR hIGear,
        DWORD dwGrpID,
        DWORD dwROP
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set ROP code.
dwROP	DWORD	ROP code to set.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR       hIGear;      /* HIGEAR handle of image  */
DWORD        nGrpID;      /* display group identifier  */
 ...
/* image will be XOR'd with destination contents */
IG_dspl_ROP_set( hIGear, nGrpID, SRCINVERT );
 ...
```

Remarks:

This code determines how the source image pixels are combined with the destination area pixels. The default ROP is SRCCOPY, which overwrites the destination area with the source image. Other codes are described in the documentation for the Windows GDI function BitBlt.

1.3.1.2.8.55 IG_dspl_scroll_get

This function calculates and returns the current scroll parameters.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_scroll_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [OUT] LPAT_MODE lpnScrollMode,
        [OUT] LPAT_SCROLL_INFO lpScrollInfo
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group from which to retrieve scroll parameters.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
lpnScrollMode	LPAT_MODE	Where ScrollbarMode is to be received. If NULL, then this parameter is ignored.
lpScrollInfo	LPAT_SCROLL_INFO	Where to copy the current scroll parameters. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
NSView* nsView = self;

HIGEAR hIGear; /* HIGEAR handle of image */
DWORD nGrpID; /* display group identifier */
AT_MODE nScrollMode; /* scroll mode */
AT_SCROLL_INFO ScrollInfo; /* scroll info */
...
IG_dspl_scroll_get( hIGear, nGrpID, (__bridge HWND)nsView, &nScrollMode, &ScrollInfo );
...
```

1.3.1.2.8.56 IG_dspl_scroll_handle

This function is designed to handle scrollbar messages from an hWnd window.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_scroll_handle(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] AT_MODE nScrlType,
        [IN] AT_MODE nScrlMode,
        [IN] LONG lScrlValue,
        [OUT] LPAT_SCROLL_INFO lpScrollInfo
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group that will be used to perform operation.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is displayed and where scrolling is performed. Pointer must be casted to non-retainable HWND type type with (__bridge HWND) operator.
nScrlType	AT_MODE	Scroll command. Its value is platform dependent. For Mac platforms, valid values are: • IG_DSPL_SCROLL_HORIZONTAL • IG_DSPL_SCROLL_VERTICAL • IG_DSPL_SCROLL_HORIZONTAL | IG_DSPL_SCROLL_VERTICAL
nScrlMode	AT_MODE	Scroll command. Its value is platform dependent. For Mac platforms it should be 0.
nScrlValue	LONG	Scroll value. It is also platform dependent and its value may have a different meaning from nScrlType. Under the Mac platform, it is not used because the scroll position is obtained directly from the scrollbars.
lpScrollInfo	LPAT_SCROLL_INFO	Receive updated scroll parameters (not scrollbar). If NULL then it is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
//  MainScrollView.h
#import <Cocoa/Cocoa.h>

@interface MainScrollView : NSScrollView
@end

//  MainScrollView.m
#import "MainScrollView.h"
```

```
#import <ImageGear18/gear.h>
…
//  Register scrolling notification
- (void)awakeFromNib
{
    [[self contentView] setPostsBoundsChangedNotifications: YES];
    NSNotificationCenter *center = [NSNotificationCenter defaultCenter] ;
    [center addObserver: self
               selector: @selector(boundsDidChangeNotification:)
                   name: NSViewBoundsDidChangeNotification
                 object: [self contentView]];
}
//  Handle scrolling notification
- (void) boundsDidChangeNotification: (NSNotification *) notification
{
    IG_dspl_scroll_handle(hIGear, 0, (__bridge HWND)self, IG_DSPL_SCROLL_HORIZONTAL |
IG_DSPL_SCROLL_VERTICAL, 0, 0, NULL);
    [self setNeedsDisplay: YES];
 }
```

Remarks:

This function should not be used to set the absolute scroll position; function IG_dspl_scroll_to() should be used instead.

1.3.1.2.8.57 IG_dspl_scroll_set

This function sets the scroll parameters (not scroll position) and allows you to enable and disable the vertical and horizontal scrollbars associated with a given window.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_scroll_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] AT_MODE nScrollMode,
        [IN] INT nXPage,
        [IN] INT nYPage,
        [OUT] LPAT_SCROLL_INFO lpScrollInfo
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set scroll options.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
nScrollMode	AT_MODE	New value of ScrollbarMode to set. Possible value is a combination of the horizontal scrollbar flag: ● IG_DSPL_HSCROLLBAR_AUTO ● IG_DSPL_HSCROLLBAR_ENABLE ● IG_DSPL_HSCROLLBAR_DISABLE and the vertical: ● IG_DSPL_VSCROLLBAR_AUTO ● IG_DSPL_VSCROLLBAR_ENABLE ● IG_DSPL_VSCROLLBAR_DISABLE
nXPage	INT	New value of scrolling page width. If 0 then it will be calculated from ClipRect.
nYPage	INT	New value of scrolling page height. If 0 then it will be calculated from ClipRect.
lpScrollInfo	LPAT_SCROLL_INFO	Where to copy new scroll parameters. If NULL then the parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
NSView* nsView = self;HIGEAR hIGear; /* HIGEAR handle of image */
DWORD nGrpID; /* display group identifier */
AT_SCROLL_INFO ScrollInfo; /* scroll info */
...
/* always hide both scrollbars */
IG_dspl_scroll_set( hIGear, nGrpID, (__bridge HWND)nsView,
IG_DSPL_HSCROLLBAR_DISABLE|IG_DSPL_VSCROLLBAR_DISABLE, 0, 0, &ScrollInfo );
...
```

1.3.1.2.8.58 IG_dspl_scroll_to

This function scrolls the image to a specified position and updates the window's scroll bars accordingly.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_scroll_to(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] INT nXPos,
        [IN] INT nYPos,
        [OUT] LPAT_SCROLL_INFO lpScrollInfo
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group where to work.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn and scrolled. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
nXPos	INT	New horizontal scroll position. Should be in a valid range that can be retrieved using the IG_dspl_scroll_get() function.
nYPos	INT	New vertical scroll position. Should be in a valid range that can be retrieved using the IG_dspl_scroll_get() function.
lpScrollInfo	LPAT_SCROLL_INFO	Where to copy the new scroll parameters. If NULL then the parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
NSView* nsView = self;
HIGEAR hIGear; /* HIGEAR handle of image */
DWORD nGrpID; /* display group identifier */
AT_SCROLL_INFO ScrollInfo; /* scroll info */
...
/* scroll down and right from current position */
IG_dspl_scroll_get( hIGear, nGrpID, (__bridge HWND)nsView, NULL, &ScrollInfo );
If( (ScrollInfo.h_cur_pos<ScrollInfo.h_max) && (ScrollInfo.v_cur_pos<ScrollInfo.v_max) )
IG_dspl_scroll_to( hIGear, nGrpID, (__bridge HWND)nsView, ScrollInfo.h_cur_pos+1,
ScrollInfo.v_cur_pos+1, NULL );
```

Remarks:

This function can be used even if a window's scroll bars are disabled. Both scroll positions should be in valid scroll range but if not, then the nearest valid value will be assigned.

1.3.1.2.8.59 IG_dspl_scroll_to_ex

This function scrolls the image to a specified position and updates the window's scroll bars accordingly.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_scroll_to_ex(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] INT nXPos,
        [IN] INT nYPos,
        [OUT] LPAT_SCROLL_INFO lpScrollInfo
        [IN] BOOL bRepaint
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group where to work.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn and scrolled. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
nXPos	INT	New horizontal scroll position. Should be in a valid range that can be retrieved using the IG_dspl_scroll_get() function.
nYPos	INT	New vertical scroll position. Should be in a valid range that can be retrieved using the IG_dspl_scroll_get() function.
lpScrollInfo	LPAT_SCROLL_INFO	Where to copy the new scroll parameters. If NULL then the parameter is ignored.
bRepaint	BOOL	This parameter determines if the function repaints scrolling content (TRUE) or not (FALSE).

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

It can be used even if a window's scroll bars are disabled. Both scroll positions should be in valid scroll range but if not, then the nearest valid value will be assigned.

This function is the same as IG_dspl_scroll_to() function, but has bRepaint parameter that when FALSE allows do not repaint the scrolling content.

1.3.1.2.8.60 IG_dspl_transparency_get

This function returns the transparency parameters.

Declaration:

```
AT_ERRCOUNT    ACCUAPI  IG_dspl_transparency_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [OUT] LPAT_MODE lpnTranspMode,
        [OUT] LPAT_RGB lpTranspColor,
        [OUT] LPHIGEAR lphIMask,
        [OUT] LPAT_RECTANGLE lpMaskRect,
        [OUT] LPAT_POINT lpMaskLocation
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group to use.
lpnTranspMode	LPAT_MODE	Pointer where TranspMode options are returned. If NULL, then this parameter is ignored.
lpTranspColor	LPAT_RGB	Pointer where TranspColor option is returned. If NULL, then this parameter is ignored.
hphIMask	LPHIGEAR	Pointer where TranspMask option is returned. If NULL, then this parameter is ignored. Please note that this function does not change or delete the value of TranspMask, but only makes its copy.
lpMaskRect	LPAT_RECTANGLE	Pointer to where to return the value of the MaskRect option. If NULL, then this parameter is ignored. If an empty rectangle is returned then it is initialized with a rectangle equal to the entire mask image.
lpMaskLocation	LPAT_POINT	Pointer to where to return the value of MaskLocation option. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;        /* HIGEAR handle of image    */
DWORD           nGrpID;         /* display group identifier */
AT_MODE                 nTranspMode;     /* transparency mode  */
AT_RGB          TranspColor;    /* transparency color  */
HIGEAR          hIMask;         /* transparent mask    */
AT_RECTANGLE            MaskRect;       /* mask rectangle      */
AT_POINT                MaskLocation;    /* mask location   */
 ...
IG_dspl_transparency_get( hIGear, nGrpID, &nTranspMode, &TranspColor, &hIMask, &MaskRect,
&MaskLocation );
if( IG_image_is_valid( hIMask ) )
        IG_image_delete( hIMask );
 ...
```

1.3.1.2.8.61 IG_dspl_transparency_set

This function sets the transparency options.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_dspl_transparency_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nTranspMode,
        [IN] const LPAT_RGB lpTranspColor,
        [IN] HIGEAR hIMask,
        [IN] const LPAT_RECTANGLE lpMaskRect,
        [IN] const LPAT_POINT lpMaskLocation
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of the group in which to set transparency options.
nTranspMode	AT_MODE	New value of TranspMode to set. Possible value may be: ● IG_DSPL_TRANSPARENCY_NONE, or a combination of one of the following flags: ● IG_DSPL_TRANSPARENCY_COLOR ● IG_DSPL_TRANSPARENCY_MASK ● IG_DSPL_TRANSPMASK_STRETCH_TO_IMAGE and one of these flags: ● IG_DSPL_TRANSPMASK_LOCATE_TO_IMAGE ● IG_DSPL_TRANSPMASK_LOCATE_TO_CLIPRECT ● IG_DSPL_TRANSPMASK_LOCATE_ABSOLUTE
lpTranspColor	const LPAT_RGB	New value of TranspColor to set. If NULL, then this parameter is ignored.
hIMask	HIGEAR	New value of TranspMask option to set. Please note that the image is not deleted and the application code is responsible for removing the old value of this option.
lpMaskRect	const LPAT_RECTANGLE	New value of MaskRect option to set. If NULL, then this parameter is ignored. An empty rectangle is possible.
lpMaskLocation	const LPAT_POINT	New value of MaskLocation option to set. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;        /* HIGEAR handle of image  */
DWORD           nGrpID;        /* display group identifier  */
AT_MODE          nTranspMode;    /* transparency mode  */
AT_RGB          TranspColor;    /* transparency color */
HIGEAR          hIMask;         /* transparent mask   */
```

```
 ...
/* enable transparent color and mask with default mask rectangle and location oriented and
scaled with main image */
nTranspMode =
IG_DSPL_TRANSPARENCY_COLOR|IG_DSPL_TRANSPARENCY_MASK|IG_DSPL_TRANSPMASK_STRETCH_TO_IMAGE|
IG_DSPL_TRANSPMASK_LOCATE_TO_IMAGE;
/* set transparent color to white */
TranspColor.r = TranspColor.g = TranspColor.b = 255;
IG_dspl_transparency_set( hIGear, nGrpID, nTranspMode, &TranspColor, hIMask, NULL, NULL );
 ...
```

Remarks:

This function sets the transparency only for image displaying. For saving image with transparency mask use function IG_display_transparent_set().

1.3.1.2.8.62 IG_dspl_zoom_get

This function returns the current zoom values.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_zoom_get(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [OUT] LPAT_MODE lpnZoomMode,
        [OUT] LPDOUBLE lpdblHZoom,
        [OUT] LPDOUBLE lpdblVZoom
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of the group from which to obtain zoom parameters.
hWnd	HWND	Pointer to NSView or NSScrollView object where image is drawn. Pointer must be casted to non-retainable HWND type with (__bridge HWND) operator.
lpnZoomMode	LPAT_MODE	Pointer to where ZoomMode is to be received. If NULL, then this parameter is ignored.
lpdblHZoom	LPDOUBLE	Where to return the calculated horizontal zoom value. This value is always calculated whether or not the horizontal zoom is fixed.
lpdblVZoom	LPDOUBLE	Where to return the calculated vertical zoom value. This value is always calculated whether or not the vertical zoom is fixed.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
NSView* nsView = self;
HIGEAR hIGear; /* HIGEAR handle of image */
DWORD nGrpID; /* display group identifier */
AT_MODE nZoomMode; /* zoom mode */
DOUBLE dHZoom; /* horizontal zoom value */
DOUBLE dVZoom; /* vertical zoom value */
...
IG_dspl_zoom_get( hIGear, nGrpID, (__bridge HWND)nsView, &nZoomMode, &dHZoom, &dVZoom );
...
```

1.3.1.2.8.63 IG_dspl_zoom_set

This function sets the new zoom values.

Declaration:

```
AT_ERRCOUNT   ACCUAPI  IG_dspl_zoom_set(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] AT_MODE nZoomMode,
        [IN] DOUBLE dHZoom,
        [IN] DOUBLE dVZoom
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image handle to where set zoom options.
dwGrpID	DWORD	Identifier of group in which to set zoom options.
nZoomlMode	AT_MODE	New value of ZoomMode to set. Possible value is combination from one of the horizontal zoom flags: ● IG_DSPL_ZOOM_H_NOT_FIXED ● IG_DSPL_ZOOM_H_FIXED and one of the vertical flags: ● IG_DSPL_ZOOM_V_NOT_FIXED ● IG_DSPL_ZOOM_V_FIXED
dHZoom	DOUBLE	New value of ZoomValueH to set. Any positive number is accepted. Value 1.0 means that the image is zoomed by 100% or displayed in its actual size.
dVZoom	DOUBLE	New value of ZoomValueV to set. Any positive number is accepted. Value 1.0 means that the image is zoomed by 100% or displayed in its actual size.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR            hIGear;        /* HIGEAR handle of image    */
DWORD            nGrpID;        /* display group identifier  */
 ...
/* set aspect ratio as not fixed and zoom image by 150% and 200% for horizontal and
vertical directions respectively  */
IG_dspl_layout_set( hIGear, nGrpID, IG_ASPECT_MODE, NULL, NULL, NULL, 0, 0,
IG_DSPL_ASPECT_NOT_FIXED, 0.0 );
IG_dspl_zoom_set( hIGear, nGrpID, IG_DSPL_ZOOM_H_FIXED| IG_DSPL_ZOOM_V_FIXED, 1.5, 2.0 );
 ...
/* the same as before but only for horizontal direction; vertical direction is zoomed
according to current fit method and device rectangle */
IG_dspl_zoom_set( hIGear, nGrpID, IG_DSPL_ZOOM_H_FIXED| IG_DSPL_ZOOM_V_NOT_FIXED, 1.5, 2.0
);
 ...
```

1.3.1.2.8.64 IG_dspl_zoom_to_rect

This function calculates the zoom and scroll values, so that the specified rectangle is fitted to ClipRect.

Declaration:

```
AT_ERRCOUNT   ACCUAPI   IG_dspl_zoom_to_rect(
        [IN] HIGEAR hIGear,
        [IN] DWORD dwGrpID,
        [IN] HWND hWnd,
        [IN] const LPAT_RECTANGLE lpZoomRect,
        [OUT] LPDOUBLE lpdblHZoom,
        [OUT] LPDOUBLE lpdblVZoom
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
dwGrpID	DWORD	Identifier of group in which to set zoom_to_rect options.
hWnd	HWND	Handle of window where image is drawn.
lpZoomRect	const LPAT_RECTANGLE	Pointer to the rectangle in device coordinates that should be fitted into ClipRect.
lpdblHZoom	LPDOUBLE	Pointer to where to return the newly calculated ZoomValueH value. If NULL, then this parameter is ignored.
lpdblVZoom	LPDOUBLE	Pointer to where to return the newly calculated ZoomValueV value. If NULL, then this parameter is ignored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR           hIGear;       /* HIGEAR handle of image  */
DWORD           nGrpID;        /* display group identifier */
AT_RECTANGLE            ZoomRect;     /* rectangle where to zoom  */
RECT            rc;
 ...
GetClientRect( hWnd, &rc );
ZoomRect.x = rc.left;
ZoomRect.y = rc.top;
ZoomRect.width = (rc.right - rc.left + 1)/2;
ZoomRect.height = (rc.bottom - rc.top + 1)/2;
IG_dspl_zoom_to_rect( hIGear, nGrpID, hWnd,  &ZoomRect, NULL, NULL );
 ...
```

Remarks:

It assigns ZoomMode=IG_DSPL_ZOOM_H_FIXED| IG_DSPL_ZOOM_V_FIXED, calculates and modifies ZoomValueH, ZoomValueV, and the horizontal and vertical scroll positions. It does not change the value of rectangles DeviceRect, ImageRect and ClipRect.

1.3.1.2.9 Error Functions

This section provides information about the Error group of functions.

- IG_err_callback_get
- IG_err_callback_set
- IG_err_count_get
- IG_err_error_check
- IG_err_error_get
- IG_err_error_set
- IG_err_record_get
- IG_err_stack_clear
- IG_errmngr_callback_get
- IG_errmngr_callback_set
- IG_error_check
- IG_error_clear
- IG_error_get
- IG_error_set

1.3.1.2.9.1 IG_err_callback_get

This function obtains error stack callback data and functions that are called to signal error stack changes for the current thread.

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_callback_get(
   LPVOID FAR* lplpPrivate,
   LPFNIG_ERRSTACK_ADD FAR* lplpfnAddCB,
   LPFNIG_ERRSTACK_CLEAR FAR* lplpfnClearCB
);
```

Arguments:

Name	Type	Description
lplpPrivate	LPVOID FAR*	Pointer to LPVOID variable to retrieve the private data that is passed to *lplpfnAddCB and *lplpfnClearCB callbacks. NULL is acceptable.
lplpfnAddCB	LPFNIG_ERRSTACK_ADD FAR*	Pointer to LPFNIG_ERRSTACK_ADD variable to retrieve the callback function that is called after the record is added to the error stack. NULL is acceptable.
lplpfnClearCB	LPFNIG_ERRSTACK_CLEAR FAR*	Pointer to LPFNIG_ERRSTACK_CLEAR variable to retrieve the callback function that is called after the error stack is cleared. NULL is acceptable.

Return Value:

Returns the code of the ImageGear error that occurred during this function call. A value of zero means no errors have occurred. Errors that occurred during this function call are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
LPVOID                lpPrivate;
LPFNIG_ERRSTACK_ADD   lpfnAdd;
LPFNIG_ERRSTACK_CLEAR lpfnClear;
AT_ERRCODE iErrCode;
iErrCode = IG_err_callback_get(&lpPrivate, &lpfnAdd, &lpfnClear);
```

Remarks:

Callback data and functions can be set using the IG_err_callback_set function.

Each thread has its own independent error stack. There are two types of callbacks - local to thread and global. This API allows you to get the thread specific callbacks. Use IG_errmngr_callback_get to get the global data and callbacks.

See Also

IG_err_callback_set

1.3.1.2.9.2 IG_err_callback_set

This function sets error stack callback data and functions that are called to signal error stack changes for the current thread.

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_callback_set(
   LPVOID lpPrivate,
   LPFNIG_ERRSTACK_ADD lpfnAddCB,
   LPFNIG_ERRSTACK_CLEAR lpfnClearCB
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Any private data that will be passed to lpfnAddCB and lpfnClearCB callbacks. NULL is acceptable.
lpfnAddCB	LPFNIG_ERRSTACK_ADD	Pointer to the callback function that will be called after the record is added to the error stack. See LPFNIG_ERRSTACK_ADD for the declaration.
lpfnClearCB	LPFNIG_ERRSTACK_CLEAR	Pointer to the callback function that will be called after the error stack is cleared. See LPFNIG_ERRSTACK_CLEAR for the declaration.

Return Value:

Returns the code of the ImageGear error that occurred during this function call. A value of zero means no errors have occurred. Errors that occurred during this function call are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
VOID ACCUAPI ErrAdd(
    LPVOID lpPrivate,
    UINT nRecord,
    INT iLineNumber,
    AT_ERRCODE iCode,
    UINT nLevel,
    AT_INT lValue1,
    AT_INT lValue2,
    LPCHAR lpFileName,
    LPCHAR lpExtratext)
{
   HWND   hWnd = (HWND)lpPrivate;
   // update error window with new records
   // ...
}

VOID ACCUAPI ErrClear(
    LPVOID lpPrivate,
    UINT nRecords)
{
   HWND   hWnd = (HWND)lpPrivate;
   // remove records from error window
   // ...
}
```

```
VOID Example_IG_err_callback_set()
{
    HWND hWnd = 0; // This assuming to be a real window
    AT_ERRCODE iErrCode;
    iErrCode = IG_err_callback_set((LPVOID)hWnd, ErrAdd, ErrClear);
}
```

Remarks:

Callback data and functions can be obtained using IG_err_callback_get function.

Each thread has its own independent error stack. There are two types of callbacks - local to thread and global. This API allows you to set the thread specific callbacks. Use IG_errmngr_callback_set to set the global data and callbacks.

See Also

IG_err_callback_get

1.3.1.2.9.3 IG_err_count_get

This function returns the total number of records (errors plus warnings) on the error stack.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_err_count_get();
```

Arguments:

None

Return Value:

Returns the total number of errors and warnings on the error stack. If errors occur during this function call, the function returns (AT_ERRCOUNT)-1, but these errors are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT errCount;
errCount = IG_err_count_get();
```

See Also

IG_err_callback_get

1.3.1.2.9.4 IG_err_error_check

This function returns the number of records of the specified level on the error stack.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_err_error_check(
   UINT nLevel
);
```

Arguments:

Name	Type	Description
nLevel	UINT	Level of errors to return. 0 means critical errors (function failure), greater levels denote warnings.

Return Value:

Returns the number of records of the specified level on the error stack. If errors occur during this function call, the function returns (AT_ERRCOUNT)-1, but these errors are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT errCount;
// get number of records of level 0
errCount = IG_err_error_check(0);
```

See Also

IG_err_callback_get

1.3.1.2.9.5 IG_err_error_get

This function retrieves information about the record from error stack with the given index.

Declaration:

```
AT_BOOL ACCUAPI IG_err_error_get(
   UINT nLevel,
   UINT nIndex,
   LPCHAR lpszFileName,
   UINT nFNameSize,
   LPINT lpnLineNumber,
   LPAT_ERRCODE lpnCode,
   LPAT_INT lplValue1,
   LPAT_INT lplValue2,
   LPCHAR lpExtraText,
   UINT nETextSize
);
```

Arguments:

Name	Type	Description
nLevel	UINT	Level of errors to index. 0 means critical errors (function failure); greater levels denote warnings.
nIndex	UINT	Zero-based record index of the given level.
lpszFileName	LPCHAR	Pointer indicating where to return the source's file name where the error occurred.
nFNameSize	UINT	Size of the memory buffer lpszFileName.
lpnLineNumber	LPINT	Pointer indicating where to return the line number where the error occurred.
lpnCode	LPAT_ERRCODE	Pointer indicating where to return the error code.
lplValue1	LPAT_INT	Pointer indicating where to return the first associated long value.
lplValue2	LPAT_INT	Pointer indicating where to return the second associated long value.
lpExtraText	LPCHAR	Pointer indicating where to return the additional text description.
nETextSize	UINT	Size of the memory buffer lpExtraText.

Return Value:

Returns TRUE if the error information has been successfully retrieved; returns FALSE otherwise. Errors that occurred during this function call are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT errCount;
errCount = IG_err_count_get();
```

Remarks:

The difference between this function and IG_err_record_get is that this index exists for records with the specified level, nLevel; not for all records on the stack.

See Also

IG_err_callback_get

1.3.1.2.9.6 IG_err_error_set

This function places an error record onto the error stack.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_err_error_set(
   const LPCHAR lpFileName,
   INT iLineNumber,
   AT_ERRCODE nCode,
   UINT nLevel,
   AT_INT lplValue1,
   AT_INT lplValue2,
   const LPCHAR lpExtraText
);
```

Arguments:

Name	Type	Description
lpFileName	const LPCHAR	Pointer to a string that supplies the name of the module from which the error was generated. It is recommended that you use the _FILE_ constant in this field.
iLineNumber	INT	An integer number indicating the line from which the error was set. It is recommended that you use the _LINE_ constant in this field.
nCode	AT_ERRCODE	An integer value of type AT_ERRCODE. Set this to the code number of the error that you wish to place on the error stack.
nLevel	UINT	The level of error. 0 means critical error (function failure); greater levels denote warnings.
lplValue1	AT_INT	Two LONG arguments are available so that you may supply any supporting information about the error. Your application might use these values to decide what to do after setting a particular kind of error. This is the first one.
lplValue2	AT_INT	The second argument for the supporting information about the error. See lplValue1.
lpExtraText	const LPCHAR	Additional text description of the error. It can be NULL if it is not available.

Return Value:

Returns the code of the ImageGear error that occurred during this function call. A value of zero means no errors have occurred. Errors that occurred during this function call are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCODE iErrCode;
static const AT_ERRCODE MYERR_BAD_RASTER = (IGE_LAST_ERROR_NUMBER - 1);

// set application specific waining
iErrCode = IG_err_error_set( __FILE__, __LINE__, MYERR_BAD_RASTER, 2, 0, 0, "Some
explanation"  );
```

Remarks:

If you are setting an error code that you have defined yourself, you must make sure that it has a value less than ImageGear's IGE_LAST_ERROR_NUMBER. As the defined value of IGE_LAST_ERROR_NUMBER may change in the future, you should define your error codes relatively to IGE_LAST_ERROR_NUMBER, as demonstrated in the example, rather than use literal values.

1.3.1.2.9.7 IG_err_record_get

This function obtains information about the record with the given index from the error stack.

Declaration:

```
AT_BOOL ACCUAPI IG_err_record_get(
   UINT nIndex,
   LPCHAR lpszFileName,
   UINT nFNameSize,
   LPINT lpnLineNumber,
   LPAT_ERRCODE lpnCode,
   LPUINT lpnLevel,
   LPAT_INT lplValue1,
   LPAT_INT lplValue2,
   LPCHAR lpExtraText,
   UINT nETextSize
);
```

Arguments:

Name	Type	Description
nIndex	UINT	Zero based index of the record.
lpszFileName	LPCHAR	Pointer indicating where to return the source's file name where the error occurred.
nFNameSize	UINT	The buffer memory size of lpszFileName.
lpnLineNumber	LPINT	Pointer indicating where to return the line number where the error occurred.
lpnCode	LPAT_ERRCODE	Pointer indicating where to return the error code.
lpnLevel	LPUINT	Pointer indicating where to return the error level.
lplValue1	LPAT_INT	Pointer indicating where to return the first associated long value.
lplValue2	LPAT_INT	Pointer indicating where to return the second associated long value.
lpExtraText	LPCHAR	Pointer indicating where to return additional text description.
nETextSize	UINT	Size of the memory buffer lpExtraText.

Return Value:

Returns TRUE if the record information has been successfully retrieved; returns FALSE otherwise. Errors that occurred during this function call are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT iErrCount, i;
CHAR         FileName[_MAX_PATH];
INT          nLineNumber;
AT_ERRCODE   nCode;
UINT         nLevel;
// get all records from error stack.
iErrCount = IG_err_count_get( );
for( i = 0; i<iErrCount; i++ )
{
    IG_err_record_get( i, FileName, sizeof(FileName), &nLineNumber, &nCode, &nLevel, NULL,
NULL, NULL, 0 );
```

```
    //...
}
```

Remarks:

This index is the general index of all records on the stack. The difference between this function and IG_err_error_get is that this function enumerates all records rather than only records of a given level.

See Also

IG_err_callback_get

1.3.1.2.9.8 IG_err_stack_clear

This function removes all records from the error stack.

Declaration:

```
AT_ERRCODE ACCUAPI IG_err_stack_clear();
```

Arguments:

None

Return Value:

Returns TRUE if the error stack has been successfully cleared; returns FALSE otherwise. Errors that occurred during this function call are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
IG_err_stack_clear();
```

See Also

IG_err_callback_get

1.3.1.2.9.9 IG_errmngr_callback_get

This function obtains error stack callback data and functions that are called to signal error stack changes for all threads.

Declaration:

```
AT_ERRCODE ACCUAPI IG_errmngr_callback_get(
   LPVOID FAR* lplpPrivate,
   LPFNIG_ERRMNGR_ADD FAR* lplpfnAddCB,
   LPFNIG_ERRMNGR_CLEAR FAR* lplpfnClearCB
);
```

Arguments:

Name	Type	Description
lplpPrivate	LPVOID FAR*	Pointer to LPVOID variable to retrieve the private data that is passed to *lplpfnAddCB and *lplpfnClearCB callbacks. NULL is acceptable.
lplpfnAddCB	LPFNIG_ERRMNGR_ADD FAR*	Pointer to LPFNIG_ERRMNGR_ADD variable to retrieve the callback function that is called after the record is added to the error stack. NULL is acceptable.
lplpfnClearCB	LPFNIG_ERRMNGR_CLEAR FAR*	Pointer to LPFNIG_ERRMNGR_CLEAR variable to retrieve the callback function that is called after the error stack is cleared. NULL is acceptable.

Return Value:

Returns the code of the ImageGear error that occurred during this function call. A value of zero means no errors have occurred. Errors that occurred during this function call are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
LPVOID Private;
LPFNIG_ERRMNGR_ADD lpfnAdd;
LPFNIG_ERRMNGR_CLEAR lpfnClear;
AT_ERRCODE iErrCode;

iErrCode  = IG_errmngr_callback_get( &Private, &lpfnAdd, &lpfnClear );
```

Remarks:

Global private data and callback functions can be set using IG_errmngr_callback_set function.

Each thread has its own independent error stack. There are two types of callbacks - local to thread and global. This API allows you to get the global (thread independent) callbacks. Use IG_err_callback_get to get the thread specific data and callbacks.

See Also

IG_errmngr_callback_set

1.3.1.2.9.10 IG_errmngr_callback_set

This function sets error stack callback data and functions that are called to signal error stack changes for all threads.

Declaration:

```
AT_ERRCODE ACCUAPI IG_errmngr_callback_set(
   LPVOID lpPrivate,
   LPFNIG_ERRMNGR_ADD lpfnAddCB,
   LPFNIG_ERRMNGR_CLEAR lpfnClearCB
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Any Private data that will be passed to lpfnAddCB and lpfnClearCB callbacks. NULL is acceptable.
lpfnAddCB	LPFNIG_ERRMNGR_ADD	Callback function that will be called after the record is added to the error stack. See LPFNIG_ERRMNGR_ADD for a declaration.
lpfnClearCB	LPFNIG_ERRMNGR_CLEAR	Callback function that will be called after the error stack is cleared. See LPFNIG_ERRMNGR_CLEAR for a declaration.

Return Value:

Returns the code of the ImageGear error that occurred during this function call. A value of zero means no errors have occurred. Errors that occurred during this function call are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
VOID ACCUAPI ErrMgrRecordAdd(
   LPVOID        lpPrivate,
   DWORD            dwThreadID,
   UINT            nRecord,
    INT            iLineNumber,
    AT_ERRCODE    iCode,
    UINT            nLevel,
    AT_INT        lValue1,
    AT_INT        lValue2,
    LPCHAR        lpFileName,
    LPCHAR        lpExtratext
)
{
    HWND hWnd = (HWND)lpPrivate;
    // update error window with new records
    // ...
}

VOID ACCUAPI ErrMgrClear(
   LPVOID        lpPrivate,
   DWORD            dwThreadID,
   UINT            nRecords)
{
    HWND hWnd = (HWND)lpPrivate;
    // remove records from error window
    // ...
}

void Example_IG_errmngr_callback_set()
{
    HWND hWnd = 0; // This assuming to be a real window
    AT_ERRCODE iErrCode;

    iErrCode = IG_errmngr_callback_set( (LPVOID)hWnd, ErrMgrRecordAdd, ErrMgrClear);
}
```

Remarks:

Global private data and callback functions can be obtained using IG_errmngr_callback_get function.

Each thread has its own independent error stack. There are two types of callbacks - local to thread and global. This API allows you to get the global (thread independent) callbacks. Use IG_err_callback_set to set the thread specific data and callbacks.

1.3.1.2.9.11 IG_error_check

This function returns the number of errors currently on the ImageGear error stack.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_error_check();
```

Arguments:

None

Return Value:

Returns the number of errors on the error stack. If errors occur during this function call, the function returns (AT_ERRCOUNT)-1, but these errors are not appended onto the error stack. A value of zero means no ImageGear errors have occurred during your last IG_ function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

A call to this function has the same effect as a call to IG_err_error_check with nLevel equal to 0.

See Also

IG_error_get

IG_error_set

IG_error_clear

1.3.1.2.9.12 IG_error_clear

This function clears all errors from the error stack.

Declaration:

```
VOID ACCUAPI IG_error_clear();
```

Arguments:

None

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
BYTE szModuleName[30];
INT  iNameSize;
INT  iLineNumber;
AT_ERRCODE  iCode;
AT_ERRCOUNT nErrcount;
INT n;
iNameSize = 30;
nErrcount = IG_error_check();
for ( n = 0; n < nErrcount; n++ )
{
    IG_error_get( n, (LPSTR)szModuleName, iNameSize,
        &iLineNumber, &iCode, NULL, NULL );
}
IG_error_clear();
```

Remarks:

After calling this function, IG_error_check will return zero.

1.3.1.2.9.13 IG_error_get

This function retrieves an ImageGear Error Code and associated information from the error stack.

Declaration:

```
VOID ACCUAPI IG_error_get(
   INT iErrorIndex,
   LPSTR szFileName,
   INT cbFileNameSize,
   LPINT lpiLineNumber,
   LPAT_ERRCODE lpiCode,
   LPAT_INT lplValue1,
   LPAT_INT lplValue2
);
```

Arguments:

Name	Type	Description
iErrorIndex	INT	Tells which error to fetch from stack. A value of 0 means fetch the first error placed on the stack.
szFileName	LPSTR	Pointer indicating where to return the module name in which this error occurred. If this pointer is NULL, the module name is not returned.
cbFileNameSize	INT	Number of bytes available in byte array pointed to by szFileName.
lpiLineNumber	LPINT	Pointer indicating where to return the line number at which the error occurred. If NULL, the line number is not returned.
lpiCode	LPAT_ERRCODE	Pointer indicating where to return the Error Code. If NULL, the Error Code is not returned.
lplValue1	LPAT_INT	Pointer indicating where to return a value stored as lValue1 when the error occurred. If NULL, this value is not returned. See Remarks below for explanation of lValue1 and lValue2.
lplValue2	LPAT_INT	Pointer indicating where to return a value stored as lValue2 when the error occurred. If NULL, this value is not returned. See Remarks below for explanation of lValue1 and lValue2.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
INT        i;                // Will hold Loop Index and Error Index
INT       iLineNumber;        // Will hold returned Line Number
BYTE      szFileName[30];    // Will hold ret'd module name, up to 29 chars
INT       cbFileNameSize;    // Will hold size of szFileName array
AT_INT      lValue1, lValue2;// Will hold returned lValue1, lValue2
AT_ERRCODE  iCode;             // Will hold returned ImageGear Error Code
AT_ERRCOUNT  nErrcount;          // Will hold count of errors on error stack
TCHAR szBuf[60];             // Will hold zero-terminated string returned by wsprintf()
below
cbFileNameSize = 30;         // Size of module-name array
nErrcount = IG_error_check(); // Get number of errors on stack
for ( i = 0;  i < nErrcount;  i++ )
{
    // Get Module Name, Line Number, Error Code, and lValue1, lValue2:
    IG_error_get ( i, (LPSTR) &szFileName,
            cbFileNameSize, &iLineNumber, (LPAT_ERRCODE)&iCode,
            (LPAT_INT) &lValue1, (LPAT_INT) &lValue2 );
    // Format error message in szBuf:
    wsprintf ( szBuf, _T("Error %d in Module %s at Line %d"), iCode, szFileName,
iLineNumber );
    // Display error message in a Message Box, with heading "Error" :
    MessageBox ( NULL, szBuf, _T("Error"), MB_OK );
}
IG_error_clear(); // Done getting errors, clear the error stack
```

Remarks:

Set iErrorIndex to indicate which error to get. iErrorIndex = 0 means the error added to the stack first. The other arguments (except cbFileNameSize) are pointers telling this function where to return the retrieved information to you. This information consists of the Error Code, the module name and line number at which the error occurred, and two additional values (lValue1 and lValue2) which may provide additional information about the error. See for a list of all ImageGear Error Codes and the significance of lValue1, lValue2 where applicable.

To determine the number of errors currently on the error stack use IG_error_check. After fetching all error information you need using IG_error_get, use IG_error_clear to clear the stack.

A call to this function has the same effect as a call to IG_err_error_get with nLevel equal to 0 and lpExtraText equal to NULL.

1.3.1.2.9.14 IG_error_set

This function places an ImageGear error onto the error stack.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_error_set(
   const LPSTR szFileName,
   INT iLineNumber,
   AT_ERRCODE iCode,
   AT_INT lValue1,
   AT_INT lValue2
);
```

Arguments:

Name	**Type**	**Description**
szFileName	const LPSTR	Pointer to a string that supplies the name of the module from which the error was generated. It is recommended that you use the _FILE_ constant in this field.
iLineNumber	INT	An integer telling ImageGear from which line the error was set. It is recommended that you use the _LINE_ constant in this field.
iCode	AT_ERRCODE	An integer value of type AT_ERRCODE. Set this to the code number of the error that you wish to place on the error stack.
lValue1	AT_INT	The first argument that supplies any supporting information about the error. Your application might use this value to decide what to do after setting a particular kind of error.
lValue2	AT_INT	The second argument that supplies any supporting information about the error. Your application might use this value to decide what to do after setting a particular kind of error.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
static const AT_ERRCODE MYERR_BAD_RASTER = (IGE_LAST_ERROR_NUMBER - 1);
AT_ERRCOUNT nErrcount = IG_error_set(__FILE__, __LINE__, MYERR_BAD_RASTER, 0, 0);
```

Remarks:

One use for this function is with the callback functions. It allows you to write the loading and saving of DIBs and individual raster lines (IG_load_FD_CB, IG_save_FD_CB_ex) to the handle. Each of these functions call your callback functions in order to supply data, such as width, height, and Bits Per Pixel, to ImageGear. Your callback functions (which must be of type LPFNIG_DIB_GET, LPFNIG_RASTER_GET, LPFNIG_DIB_CREATE, and LPFNIG_RASTER_SET), must return an error count to IG_load_FD_CB and IG_save_FD_CB. You would do this by calling IG_error_check after each raster is loaded or saved. If you wanted to terminate the load or save, you could use IG_error_set to place the ImageGear error of your choice upon the stack.

If you are setting an error code that you have defined yourself, you must make sure that it has a value less than ImageGear's IGE_LAST_ERROR_NUMBER. As the defined value of IGE_LAST_ERROR_NUMBER may change in the future, you should define your error codes relatively to IGE_LAST_ERROR_NUMBER, as demonstrated in the example, rather than use literal values.

1.3.1.2.10 Filter Functions

This section provides information about the Filter group of functions.

- IG_fltr_compressionlist_get
- IG_fltr_compressionlist_get_ex
- IG_fltr_ctrl_get
- IG_fltr_ctrl_list
- IG_fltr_ctrl_set
- IG_fltr_detect_FD
- IG_fltr_detect_file
- IG_fltr_detect_get
- IG_fltr_detect_mem
- IG_fltr_detect_set
- IG_fltr_formatlist_get
- IG_fltr_formatlist_sort
- IG_fltr_ICC_callback_get
- IG_fltr_ICC_callback_set
- IG_fltr_info_get
- IG_fltr_load_FD_format
- IG_fltr_load_file
- IG_fltr_load_file_format
- IG_fltr_metad_callback_get
- IG_fltr_metad_callback_set
- IG_fltr_metad_update_file
- IG_fltr_pagecount_FD_format
- IG_fltr_pagecount_file_format
- IG_fltr_pagedelete_file
- IG_fltr_pageinfo_get
- IG_fltr_pageinfo_get_ex
- IG_fltr_pageswap_file
- IG_fltr_raster_plane_callback_get
- IG_fltr_raster_plane_callback_set
- IG_fltr_save_FD_size_calc
- IG_fltr_save_file
- IG_fltr_save_file_size_calc
- IG_fltr_save_mem
- IG_fltr_save_mem_size_calc
- IG_fltr_savelist_get
- IG_fltr_savelist_get_ex

1.3.1.2.10.1 IG_fltr_compressionlist_get

This function returns the list of compressions available for saving the specified image to a particular file format. This is an obsolete function, see remarks.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_compressionlist_get(
   LPAT_DIB lpDIB,
   AT_MODE nFormatID,
   LPAT_MODE lpComprList,
   UINT nCListSize,
   LPUINT lpnCListCount
);
```

Arguments:

Name	Type	Description
lpDIB	LPAT_DIB	Pointer to the AT_DIB structure that contains image parameters. If NULL, then this function returns all possible compressions.
nFormatID	AT_MODE	File format identifier. See enumIGFormats for possible values.
lpComprList	LPAT_MODE	Pointer to the array to return the list of compression constants to. See enumIGCompressions for possible values. Set to NULL if you only need to obtain the number of available compressions.
nCListSize	UINT	Size of the lpCompList array.
lpnCListCount	LPUINT	Pointer to a variable which will receive the actual number of compression constants.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

Example:

```
AT_ERRCOUNT nErrCount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
UINT nCount, nActual;   // Number of compressions
LPAT_MODE lpCompList;   // List of compressions

// Load the image
nErrCount = IG_load_file("picture.tif", &hIGear);
if( nErrCount == 0 )
{
    AT_DIB atDib;
    AT_DIMENSION nWidth, nHeight;
    UINT nBitsPerPixel;
    // Get image info
    nErrCount = IG_image_dimensions_get(hIGear, &nWidth, &nHeight, &nBitsPerPixel);
    // Fill in AT_DIB structure
    memset(&atDib, 0, sizeof(AT_DIB));
    atDib.biSize = sizeof(AT_DIB);
    atDib.biWidth = nWidth;
    atDib.biHeight = nHeight;
    atDib.biPlanes = 1;
    atDib.biBitCount = nBitsPerPixel;

    // Get compression count
    nErrCount = IG_fltr_compressionlist_get(&atDib,
        IG_FORMAT_TIF, NULL, 0, &nCount);
    // Allocate memory for compressions
    lpCompList = (LPAT_MODE) malloc(nCount * sizeof(AT_MODE));
    // Get list of compressions that can be used when
    // saving the given image into TIFF format
    nErrCount = IG_fltr_compressionlist_get(&atDib,
        IG_FORMAT_TIF, lpCompList, nCount, &nActual);

    // ...

    // Delete memory
    free(lpCompList);
    // Delete the image
    IG_image_delete(hIGear);
}
```

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_fltr_compressionlist_get_ex instead.

See also the section Using Format Filters API for Image Saving.

1.3.1.2.10.2 IG_fltr_compressionlist_get_ex

This function returns the list of compressions available for saving the specified image to a particular file format.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_compressionlist_get_ex(
   const HIGDIBINFO hDIB,
   AT_MODE nFormatID,
   LPAT_MODE lpComprList,
   UINT nCListSize,
   LPUINT lpnCListCount
);
```

Arguments:

Name	Type	Description
hDIB	const HIGDIBINFO	Handle of DIB info object that contains image parameters. If NULL, then this function returns all possible compressions.
nFormatID	AT_MODE	File format identifier. See enumIGFormats for possible values.
lpComprList	LPAT_MODE	Pointer to the array to return the list of compression constants to. See enumIGCompressions for possible values. Set to NULL if you only need to obtain the number of available compressions.
nCListSize	UINT	Size of the lpCompList array.
lpnCListCount	LPUINT	Pointer to a variable which will receive the actual number of compression constants.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGearProfessional.

Example:

```
AT_ERRCOUNT nErrCount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
HIGDIBINFO hDIB;        // DIB info handle of image
UINT nCount, nActual;   // Number of compressions
LPAT_MODE lpCompList;   // List of compressions

// Load the image
nErrCount = IG_load_file("picture.tif", &hIGear);
if( nErrCount==0 )
{
    nErrCount = IG_image_DIB_info_get(hIGear, &hDIB);
    // Get compression count
    nErrCount = IG_fltr_compressionlist_get_ex(hDIB,
        IG_FORMAT_TIF, NULL, 0, &nCount);
    // Allocate memory for compressions
    lpCompList = (LPAT_MODE) malloc(nCount * sizeof(AT_MODE));
    // Get list of compressions that can be used when
    // saving the given image into TIFF format
    nErrCount = IG_fltr_compressionlist_get_ex(hDIB,
        IG_FORMAT_TIF, lpCompList, nCount, &nActual);

    // ...
    // Delete memory
    free(lpCompList);
    // Delete DIB info
    IG_DIB_info_delete(hDIB);
    // Delete the image
    IG_image_delete(hIGear);
}
```

Remarks:

See also the section Using Format Filters API for Image Saving.

1.3.1.2.10.3 IG_fltr_ctrl_get

This function allows you to get full information about a control parameter supported by an ImageGear filter. It also returns the current or default value of the parameter.

Declaration:

```
AT_ERRCODE ACCUAPI IG_fltr_ctrl_get(
   DWORD dwFormatID,
   const LPCHAR lpcsCtrlName,
   AT_BOOL bGetDefault,
   LPAT_MODE lpnValueType,
   LPDWORD lpdwValueSize,
   LPVOID lpBuffer,
   DWORD dwBufferSize
);
```

Arguments:

Name	Type	Description
dwFormatID	DWORD	A constant indicating the format filter for which the information should be retrived. See enumIGFormats for possible values.
lpcsCtrlName	const LPCHAR	Specifies the name of the control parameter you want to get the info about. Use IG_fltr_ctrl_list to obtain the names of control parameters supported for a format filter.
bGetDefault	AT_BOOL	Set to TRUE to obtain the default value of the control parameter. Set to FALSE to obtain the current value of the control parameter.
lpnValueType	LPAT_MODE	Returns the value type of control parameter.
lpdwValueSize	LPDWORD	Returns the size in bytes of the memory buffer that should be allocated for lpBuffer to completely fit the returned value. If NULL then this parameter is ignored.
lpBuffer	LPVOID	Pointer to a memory buffer which will be overwritten with the control parameter value.
dwBufferSize	DWORD	The size of memory allocated for lpBuffer pointer. If it is less than the value returned through lpdwValueSize then only dwBufferSize bytes will be written to the lpBuffer, and no error will be returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;            // Count of returned errors on stack
AT_MODE valueType;
DWORD valueSize;
DWORD bufferSize;
// Get a type and a size of the control parameter
nErrcount = IG_fltr_ctrl_get(IG_FORMAT_TIF, "BUFFER_SIZE", FALSE, &valueType, &valueSize,
NULL, 0);
if(nErrcount == 0)
{
    // Get the control parameter
    if(valueType == AM_TID_DWORD && valueSize == sizeof(DWORD))
    {
        nErrcount = IG_fltr_ctrl_get(IG_FORMAT_TIF, "BUFFER_SIZE", FALSE, NULL, NULL, 
&bufferSize, sizeof(bufferSize));

        // ...
    }
}
```

Remarks:

The application is responsible for allocating memory for lpBuffer and freeing it when it is no longer in use.

Use function IG_fltr_ctrl_set to change the value of a control parameter .

See also the section Using Format Filters API for Filter Control.

1.3.1.2.10.4 IG_fltr_ctrl_list

This function allows you to get the list of control parameters for each ImageGear filter you specify using FormatID argument.

Declaration:

```
AT_ERRCODE ACCUAPI IG_fltr_ctrl_list(
   DWORD dwFormatID,
   LPUINT lpnCount,
   LPVOID lpArray,
   DWORD dwArraySizeInBytes
);
```

Arguments:

Name	Type	Description
dwFormatID	DWORD	A constant indicating the format filter for which the information should be retrived. See enumIGFormats for possible values.
lpnCount	LPUINT	Returns the number of supported control parameters for the requested filter.
lpArray	LPVOID	Returns the array of control parameters for the requested filter. Application is responsible to allocate memory for lpArray buffer but IG fills each element of this array with pointer to static string. Application is not allowed to change memory referenced by any of those string pointers.
dwArraySizeInBytes	DWORD	Provides the size (in bytes) of memory allocated for lpArray pointer.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;             // Count of returned errors on stack
UINT listLength;
LPSTR* lpList;
// Get a length of the control parameter list
nErrcount = IG_fltr_ctrl_list(IG_FORMAT_TIF, &listLength, NULL, 0);
if(nErrcount == 0)
{
    // Allocate memory for the list
    lpList = (LPSTR*)malloc(listLength * sizeof(LPSTR));
    // Get the control parameter list
    nErrcount = IG_fltr_ctrl_list(IG_FORMAT_TIF, &listLength, lpList, listLength *
sizeof(LPSTR));

    // ...

    free(lpList);
}
```

Remarks:

See also the section Using Format Filters API for Filter Control.

1.3.1.2.10.5 IG_fltr_ctrl_set

This function allows you to set a control parameter value for the specified format filter.

Declaration:

```
AT_ERRCODE ACCUAPI IG_fltr_ctrl_set(
   DWORD dwFormatID,
   const LPCHAR lpcsCtrlName,
   LPVOID lpValue,
   DWORD dwValueSize
);
```

Arguments:

Name	Type	Description
dwFormatID	DWORD	A constant indicating the format filter for which the control parameter should be set. See enumIGFormats for possible values.
lpcsCtrlName	const LPCHAR	Specifies the name of control parameter you want to set. The list of names of supported control parameters can be obtained using IG_fltr_ctrl_list.
lpValue	LPVOID	Specifies the new value for the control parameter. See Remarks.
dwValueSize	DWORD	Specifies the size (in bytes) of the control parameter value.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;            // Count of returned errors on stack
DWORD bufferSize = 32767;
// Get a type and a size of the control parameter
nErrcount = IG_fltr_ctrl_set(IG_FORMAT_TIF, "BUFFER_SIZE", (LPVOID)(AT_UINT)bufferSize,
sizeof(bufferSize));
```

Remarks:

See ImageGear Supported File Formats Reference section for description of all control parameters supported by ImageGear file format filters.

Use IG_fltr_ctrl_list to get the list of supported control parameters for a specific format. Use IG_fltr_ctrl_get to get the information about a specific control parameter, as well as its current value.

The rules for passing values to this function are as follows:

- For platform-dependent integers AT_INT and AT_UINT, as well as types derived from them, such as AT_DIMENSION, lpValue should contain the actual value.
- For other types, if the value size is less than or equal to sizeof(DWORD), then lpValue should contain the actual value; otherwise it should contain a pointer to the value.

See also the section Using Format Filters API for Filter Control.

1.3.1.2.10.6 IG_fltr_detect_FD

This function detects the format of the image file specified by a file descriptor.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_detect_FD(
   AT_INT fd,
   LONG lOffset,
   LPAT_MODE lpFileType
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file containing the image. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. lOffset is usually 0.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values. If the file format can not be detected, this parameter will return IG_FORMAT_UNKNOWN.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrCount;    // will hold returned error count
AT_MODE nFormatID;
HANDLE      fd;                             //File Descriptor

fd = CreateFile(_T("picture.tif"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if(fd != INVALID_HANDLE_VALUE)
{
    nErrCount = IG_fltr_detect_FD((AT_INT)fd, 0, &nFormatID);
    if(nFormatID == IG_FORMAT_TIF)
    {
        // ...
    }
    CloseHandle(fd);
}
```

Remarks:

See also the section Detecting Image File Format.

1.3.1.2.10.7 IG_fltr_detect_file

This function detects the format of the image file specified by a filename.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_detect_file(
   const LPSTR lpszFileName,
   LPAT_MODE lpFileType
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	File name of the image which format you wish to detect.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values. If the file format can not be detected, this parameter will return IG_FORMAT_UNKNOWN.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrCount; // will hold returned error count
AT_MODE nFormatID;

nErrCount = IG_fltr_detect_file("picture.tif", &nFormatID);
if(nFormatID == IG_FORMAT_TIF)
{
    // ...
}
```

Remarks:

See also the section Detecting Image File Format.

1.3.1.2.10.8 IG_fltr_detect_get

This function checks whether the detection is enabled for the specified format and returns the format's detection priority.

Declaration:

```
AT_ERRCODE ACCUAPI IG_fltr_detect_get(
   DWORD dwFormatID,
   LPAT_BOOL lpDetectEnable,
   LPLONG lpDetectPriority
);
```

Arguments:

Name	Type	Description
dwFormatID	DWORD	Specifies the format filter ID for which the detection flag and detection priority is retrieved. See enumIGFormats for possible values.
lpDetectEnable	LPAT_BOOL	Pointer to a variable that receives the flag specifying whether or not the detection is enabled for the given format.
lpDetectPriority	LPLONG	Pointer to a variable that receives the filter detection priority. The greater this value the higher the priority.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT     nErrcount;            // Count of returned errors on stack
AT_BOOL bDetectEnable;
LONG nDetectPriority;
nErrcount = IG_fltr_detect_get(IG_FORMAT_TIF, &bDetectEnable, &nDetectPriority);
```

1.3.1.2.10.9 IG_fltr_detect_mem

This function detects the format of an image located in the memory.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_detect_mem(
   LPVOID lpImage,
   AT_UINT nSize,
   LPAT_MODE lpFileType
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to a memory buffer containing the image.
nSize	AT_UINT	Size of image in memory.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values. If the file format can not be detected, this parameter will return IG_FORMAT_UNKNOWN.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrCount; // will hold returned error count
AT_MODE nFormatID;
AT_BYTE* lpImage = NULL;

// Read an image into the memory
FILE* fp;
fopen_s(&fp, "picture.tif", "rb");
if(fp != NULL)
{
    long fileSize;
    fseek(fp, 0, SEEK_END);
    fileSize = ftell(fp);
    fseek(fp, 0, SEEK_SET);
    // Allocate memory buffer
    lpImage = (AT_BYTE*)malloc(fileSize);
    if(lpImage != NULL)
    {
        // Read file into the memory
        fread(lpImage, 1, fileSize, fp);
        // Detect file format in the memory
        nErrCount = IG_fltr_detect_mem(lpImage, fileSize, &nFormatID);
        if(nFormatID == IG_FORMAT_TIF)
        {
            // ...
        }
        // Delete memory
        free(lpImage);
    }
    fclose(fp);
}
```

Remarks:

See also the section Detecting Image File Format.

1.3.1.2.10.10 IG_fltr_detect_set

This function turns on or off the format detection procedure for the specified file format. It also sets the format detection priority.

Declaration:

```
AT_ERRCODE ACCUAPI IG_fltr_detect_set(
   DWORD dwFormatID,
   AT_BOOL bDetectEnable,
   LONG nDetectPriority
);
```

Arguments:

Name	Type	Description
dwFormatID	DWORD	Specifies the format filter ID for which the detection flag and detection priority is set. See enumIGFormats for possible values.
bDetectEnable	AT_BOOL	TRUE turns the detection on, FALSE turns it off.
nDetectPriority	LONG	Detection priority to be used during the format detection procedure. Greater values correspond to higher priority.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT     nErrcount;            // Count of returned errors on stack
nErrcount = IG_fltr_detect_set(IG_FORMAT_TIF, TRUE, 100);
```

1.3.1.2.10.11 IG_fltr_formatlist_get

This function searches and returns the list of ImageGear supported format filters which provide the specified features.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_formatlist_get(
   DWORD dwFlags,
   LPAT_MODE lpFormatList,
   UINT nFListSize,
   LPUINT lpnFListCount
);
```

Arguments:

Name	Type	Description
dwFlags	DWORD	Specifies format filter features. Can be any combination of constants from enumIGFltrFormatFlags enumeration.
lpFormatList	LPAT_MODE	Pointer to an array of AT_MODE where the file format constants will be returned. See enumIGFormats for possible values. Set to NULL if you only need to obtain the number of format filters returned by the search.
nFListSize	UINT	Size of lpFormatList array if it is not NULL.
lpnFListCount	LPUINT	If lpFormatList is not NULL, then the number of copied format identifiers are returned. Otherwise, the total number of formats is returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrCount;     // Count of errs on stack upon ret from func
UINT nCount;             // Count of supported formats
LPAT_MODE lpFormatList; // list of formats
// Get list of filters that support detect and saving:
// Get total count
nErrCount = IG_fltr_formatlist_get(IG_FLTR_DETECTSUPPORT|IG_FLTR_PAGEINSERTSUPPORT, NULL,
    0, &nCount );
if( nErrCount==0 )
{
    // Allocate memory
    lpFormatList = (LPAT_MODE)malloc( nCount*sizeof(AT_MODE) );
    if( lpFormatList!=NULL )
    {
        // Get supported formats
        IG_fltr_formatlist_get(IG_FLTR_DETECTSUPPORT|IG_FLTR_PAGEINSERTSUPPORT,
            lpFormatList, nCount, NULL );

        // ...

        // Release memory
        free( lpFormatList );
    }
}
```

Remarks:

This function searches for filters that support ALL requested features (rather than some of them).

Typically, you will call this function twice to obtain the list of functions. The first time you will obtain the number of values to allocate the lpFormatList array of the necessary size, and the second time you will receive the actual values.

See also the section Getting Information and Sorting Images.

1.3.1.2.10.12 IG_fltr_formatlist_sort

This function sorts the array of file format constants in alphabetic order based on the short format names.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_formatlist_sort(
   LPAT_MODE lpFormatList,
   UINT nFListSize
);
```

Arguments:

Name	Type	Description
lpFormatList	LPAT_MODE	Array of file format constants. See enumIGFormats for possible values.
nFListSize	UINT	Size of lpFormatList array.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrCount;    // Count of errs on stack upon ret from func
UINT nCount;            // Count of supported formats
LPAT_MODE lpFormatList; // list of formats
// Get list filters that support detect and saving:
// Get total count
nErrCount = IG_fltr_formatlist_get(IG_FLTR_DETECTSUPPORT|IG_FLTR_PAGEINSERTSUPPORT, NULL,
    0, &nCount );
if( nErrCount==0 )
{
    // Allocate memory
    lpFormatList = (LPAT_MODE)malloc( nCount*sizeof(AT_MODE) );
    if( lpFormatList!=NULL )
    {
        // Get supported formats
        IG_fltr_formatlist_get(IG_FLTR_DETECTSUPPORT|IG_FLTR_PAGEINSERTSUPPORT,
            lpFormatList, nCount, NULL );
        // Sort formats in alphabetic order */
        IG_fltr_formatlist_sort( lpFormatList, nCount );

        // ...

        // Delete memory
        free( lpFormatList );
    }
}
```

Remarks:

You can use this function to sort the list of formats obtained from IG_fltr_formatlist_get. Short file format names used for sorting correspond to short names returned by IG_fltr_info_get function.

See also the section Getting Information and Sorting Images.

1.3.1.2.10.13 IG_fltr_ICC_callback_get

This function returns the current settings for callbacks that are used for reading and writing ICC profiles.

Declaration:

```
AT_ERRCODE ACCUAPI IG_fltr_ICC_callback_get(
   LPVOID* lplpPrivate,
   LPAFT_IG_ICC_GET_CB* lplpfnGetCB,
   LPAFT_IG_ICC_SET_CB* lplpfnSetCB,
   LPAFT_ANY* lplpfnReserved
);
```

Arguments:

Name	Type	Description
lplpPrivate	LPVOID*	Private callback data.
lplpfnGetCB	LPAFT_IG_ICC_GET_CB*	GET callback function.
lplpfnSetCB	LPAFT_IG_ICC_SET_CB*	SET callback function.
lplpfnReserved	LPAFT_ANY*	Reserved.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;             // Count of returned errors on stack
LPVOID lpPrivate;                 // Private callback data
LPAFT_IG_ICC_GET_CB lpfnGetCB;     // GET callback function
LPAFT_IG_ICC_SET_CB lpfnSetCB;     // SET callback function

// Get ICC callback functions
nErrcount = IG_fltr_ICC_callback_get(&lpPrivate, &lpfnGetCB, &lpfnSetCB, NULL);
```

Remarks:

See IG_fltr_ICC_callback_set for reading and writing ICC profiles.

1.3.1.2.10.14 IG_fltr_ICC_callback_set

This function allows you to register your ICC callback functions.

Declaration:

```
AT_ERRCODE ACCUAPI IG_fltr_ICC_callback_set(
   LPVOID lpPrivate,
   LPAFT_IG_ICC_GET_CB lpfnGetCB,
   LPAFT_IG_ICC_SET_CB lpfnSetCB,
   LPAFT_ANY lpfnReserved
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Private callback data.
lpfnGetCB	LPAFT_IG_ICC_GET_CB	GET callback function.
lpfnSetCB	LPAFT_IG_ICC_SET_CB	SET callback function.
lpfnReserved	LPAFT_ANY	Reserved.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
VOID ACCUAPI ICCGet(
   AT_VOID       *lpPrivate,         // Private callback data.
   AT_BYTE       *lpICCData,         // ICC profile data, allocated by the toolkit
   AT_INT         DataLength         // Length of ICC profile data, in bytes
)
{
    // ...
}

VOID ACCUAPI ICCSet(
   AT_VOID       *lpPrivate,         // Private callback data.
   AT_BYTE       **lplpICCData,         // ICC profile data, allocated by the application
   AT_INT        *lpDataLength         // Length of ICC profile data, in bytes
)
{
    // ...
}

void Example_IG_fltr_ICC_callback_set()
{
    AT_ERRCOUNT nErrcount;             // Count of returned errors on stack
    // Set ICC callback functions
    nErrcount = IG_fltr_ICC_callback_set(NULL, ICCGet, ICCSet, NULL);
}
```

Remarks:

This function registers callbacks for reading and writing ICC profiles during loading and saving of image files.

- ICC profile reading. Use any image loading function to load an image. As soon as the format filter encounters an ICC profile, it calls the callback function. The ICC profile is provided in the standard ICC format, as a byte array. The toolkit owns the buffer, so the application shall not delete it. If the application needs to use the ICC profile after exiting the callback, it shall copy it to its own buffer.
- ICC profile writing. Use any image saving function to save an image. If the format filter supports ICC profile writing, it calls the callback function before writing the profile. If the HIGEAR being saved has an ICC profile attached to it, format filter ignores this profile and writes the profile obtained from the callback. The ICC profile shall be provided in the standard ICC format, as a byte array. The application owns the buffer, and is responsible for deleting it.

1.3.1.2.10.15 IG_fltr_info_get

This function returns information about the format filter and its supported features.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_info_get(
   AT_MODE nFormatID,
   LPDWORD lpdwInfoFlags,
   LPCHAR lpShortName,
   DWORD dwSNameSize,
   LPCHAR lpFullName,
   DWORD dwFNameSize,
   LPCHAR lpDefExt,
   DWORD dwDefExtSize
);
```

Arguments:

Name	Type	Description
nFormatID	AT_MODE	File format identifier. See enumIGFormats for possible values.
lpdwInfoFlags	LPDWORD	Pointer to a variable which will receive the format flags. Any combination of enumIGFltrFormatFlags values can be returned.
lpShortName	LPCHAR	Pointer to a byte array which will receive the zero-terminated string with the short file format name. Set to NULL if you do not need to obtain this information.
dwSNameSize	DWORD	Size of the lpShortName array in bytes.
lpFullName	LPCHAR	Pointer to a byte array which will receive the zero-terminated string with the full file format name. Set to NULL if you do not need to obtain this information.
dwFNameSize	DWORD	Size of the lpFullName array in bytes.
lpDefExt	LPCHAR	Pointer to a byte array which will receive the zero-terminated string with file masks such as *.tif separated by a semicolon (;). Set to NULL if you do not need to obtain this information.
dwDefExtSize	DWORD	Size of lpDefExt array in bytes.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_CHAR shortName[_MAX_PATH];
AT_CHAR fullName[_MAX_PATH];
AT_CHAR defExt[_MAX_PATH];
DWORD dwFlags;

AT_ERRCOUNT nErrCount;
nErrCount = IG_fltr_info_get(IG_FORMAT_TIF, &dwFlags, shortName, sizeof(shortName),
    fullName, sizeof(fullName), defExt, sizeof(defExt));

// Output the filter info
if(dwFlags & IG_FLTR_DETECTSUPPORT)
    printf("IG_FLTR_DETECTSUPPORT\n");
if(dwFlags & IG_FLTR_PAGEREADSUPPORT)
```

```
    printf("IG_FLTR_PAGEREADSUPPORT\n");
if(dwFlags & IG_FLTR_MPAGEREADPSUPPORT)
    printf("IG_FLTR_MPAGEREADPSUPPORT\n");
if(dwFlags & IG_FLTR_MPAGEWRITEPSUPPORT)
    printf("IG_FLTR_MPAGEWRITEPSUPPORT\n");
if(dwFlags & IG_FLTR_PAGEINSERTSUPPORT)
    printf("IG_FLTR_PAGEINSERTSUPPORT\n");
if(dwFlags & IG_FLTR_PAGEDELETESUPPORT)
    printf("IG_FLTR_PAGEDELETESUPPORT\n");
if(dwFlags & IG_FLTR_PAGESWAPSUPPORT)
    printf("IG_FLTR_PAGESWAPSUPPORT\n");
if(dwFlags & IG_FLTR_MPDATASUPPORT)
    printf("IG_FLTR_MPDATASUPPORT\n");

printf("Short name: %s\nFullName: %s\nDefault Extension: %s\n", shortName, fullName,
defExt);
```

Remarks:

This function returns a short format name - usually 3-4 chars (e.g., "TIFF"), full format name (e.g., "Tagged Image File Format"), and default file extensions separated by ";" (e.g., "*.tif;*.tiff").

See also the section Getting Information and Sorting Images.

1.3.1.2.10.16 IG_fltr_load_FD_format

This function loads an image from an open file into memory and creates a HIGEAR handle for the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_load_FD_format(
   AT_MODE nFormat,
   AT_INT fd,
   LONG lOffset,
   UINT nPage,
   UINT nTile,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
nFormat	AT_MODE	A constant indicating the file format of the input file. See enumIGFormats for possible values. Set to IG_FORMAT_UNKNOWN to let ImageGear detect the file format.
fd	AT_INT	Handle of the open file containing the image. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset to image in the file.
nPage	UINT	Page number to load if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
nTile	UINT	Tile number to load, set to 1 for non-tiled image.
lphIGear	LPHIGEAR	ImageGear handle returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR    hIGear;        // Will hold handle returned by IG_fltr_load_file
AT_ERRCOUNT nErrCount;    // Count of errs on stack upon ret from func*/
HANDLE      fd;            //File Descriptor
fd = CreateFile(_T("picture.tif"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if(fd != INVALID_HANDLE_VALUE)
{
    // Load the selected image
    nErrCount = IG_fltr_load_FD_format(IG_FORMAT_TIF, (AT_INT)fd, 0, 1, 0, &hIGear);
    CloseHandle(fd);
    if(nErrCount == 0)
    {
        // ...

        // Delete the image
        IG_image_delete(hIGear);
    }
}
```

Remarks:

If nFormat = IG_FORMAT_UNKNOWN then ImageGear attempts to detect the file format automatically, and then loads the image. Otherwise, ImageGear skips the file format detection and loads the file with the specified format filter.

See also the section Getting Information about a File Format Filter.

1.3.1.2.10.17 IG_fltr_load_file

This function loads an image from the specified file into memory and creates a HIGEAR handle for this image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_load_file(
   const LPSTR lpszFileName,
   UINT nPage,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Path and name of the file to load. The path can be absolute or relative.
nPage	UINT	Page number to load if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
lphIGear	LPHIGEAR	Pointer to the HIGEAR object in which to return the ImageGear handle of the image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

The handle that ImageGear assigns for this image is returned in the hIGear argument. The file named by szFileName may be in any format recognized by ImageGear. The function will determine the format by inspecting the file's header section.

Example:

```
HIGEAR    hIGear;         // Will hold handle returned by IG_fltr_load_file
AT_ERRCOUNT nErrCount;     // Count of errs on stack upon ret from func*/
// Load the selected image
nErrCount = IG_fltr_load_file("picture.tif", 1, &hIGear);
if(nErrCount == 0)
{
    // ...

    // Delete the image
    IG_image_delete(hIGear);
}
```

- Some file formats, such as TXT (ASCII Text), JPEG, and others, may be loaded with additional control, using IG_fltr_ctrl_get and IG_fltr_ctrl_set. See the description of these functions also in Using Format Filters API for Filter Control section.
- Note that simply loading the file does not cause it to be displayed. Refer to IG_dspl_image_draw and related routines, for how to display an image once it is in memory. See also IG_load_file_display.

See also the section Loading Images.

1.3.1.2.10.18 IG_fltr_load_file_format

This function loads an image from the specified file into memory and creates a HIGEAR handle for this image. The function allows to skip the automatic detection of the file format and instead use the specified format ID.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_load_file_format(
   AT_MODE nFormat,
   const LPSTR lpszFileName,
   UINT nPage,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
nFormat	AT_MODE	A constant indicating the file format of the input file. See enumIGFormats for possible values. Set to IG_FORMAT_UNKNOWN to let ImageGear detect the file format.
lpszFileName	const LPSTR	Path and name of the file to load. The path can be absolute or relative.
nPage	UINT	Page number to load if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
lphIGear	LPHIGEAR	Pointer to the HIGEAR object in which to return the ImageGear handle of the image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR    hIGear;         // Will hold handle returned by IG_fltr_load_file
AT_ERRCOUNT nErrCount;     // Count of errs on stack upon ret from func*/
// Load the selected image
nErrCount = IG_fltr_load_file_format(IG_FORMAT_TIF, "picture.tif", 1, &hIGear);
if(nErrCount == 0)
{
    // ...

    // Delete the image
    IG_image_delete(hIGear);
}
```

Remarks:

If nFormat = IG_FORMAT_UNKNOWN then ImageGear attempts to detect the file format automatically, and then loads the image. Otherwise, ImageGear skips the file format detection and loads the file with the specified format filter.

See also the section Getting Information about a File Format Filter.

1.3.1.2.10.19 IG_fltr_metad_callback_get

This function returns the callback functions that ImageGear uses to pass or receive metadata during save and load operations.

Declaration:

```
AT_ERRCODE ACCUAPI IG_fltr_metad_callback_get(
   LPVOID* lpPrivate,
   LPAFT_IG_METAD_ITEM_SET_CB* lplpfnSetCB,
   LPAFT_IG_METAD_ITEM_ADD_CB* lplpfnAddCB,
   LPAFT_IG_METAD_ITEM_GET_CB* lplpfnGetCB
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID*	Returns private data associated with metadata callback functions.
lplpfnSetCB	LPAFT_IG_METAD_ITEM_SET_CB*	Returns pointer to callback function of type LPAFT_IG_METAD_ITEM_SET_CB that is used for Set metadata operation.
lplpfnAddCB	LPAFT_IG_METAD_ITEM_ADD_CB*	Returns pointer to callback function of type LPAFT_IG_METAD_ITEM_ADD_CB that is used for Add metadata operation.
lplpfnGetCB	LPAFT_IG_METAD_ITEM_GET_CB*	Returns pointer to callback function of type LPAFT_IG_METAD_ITEM_GET_CB that is used for Get metadata operation.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;         // Count of returned errors on stack
LPVOID lpPrivate;
LPAFT_IG_METAD_ITEM_SET_CB lpfnSetCB;
LPAFT_IG_METAD_ITEM_ADD_CB lpfnAddCB;
LPAFT_IG_METAD_ITEM_GET_CB lpfnGetCB;

// Get metadata callback functions
nErrcount = IG_fltr_metad_callback_get(&lpPrivate, &lpfnSetCB, &lpfnAddCB, &lpfnGetCB);
```

Remarks:

A NULL value is valid for any parameter, if the corresponding information is not necessary to the application.

See also IG_fltr_metad_callback_set, LPAFT_IG_METAD_ITEM_ADD_CB, LPAFT_IG_METAD_ITEM_GET_CB, LPAFT_IG_METAD_ITEM_SET_CB functions and the section Processing of non-image data through filter callback functions.

1.3.1.2.10.20 IG_fltr_metad_callback_set

This function sets the callback functions that ImageGear uses to pass or receive metadata during save and load operations.

Declaration:

```
AT_ERRCODE ACCUAPI IG_fltr_metad_callback_set(
   LPVOID lpPrivate,
   LPAFT_IG_METAD_ITEM_SET_CB lpfnSetCB,
   LPAFT_IG_METAD_ITEM_ADD_CB lpfnAddCB,
   LPAFT_IG_METAD_ITEM_GET_CB lpfnGetCB
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	New value for private data to be associated with callback functions.
lpfnSetCB	LPAFT_IG_METAD_ITEM_SET_CB	New value of callback function of type LPAFT_IG_METAD_ITEM_SET_CB that is to be used for Set metadata operation.
lpfnAddCB	LPAFT_IG_METAD_ITEM_ADD_CB	New value of callback function of type LPAFT_IG_METAD_ITEM_ADD_CB that is to be used for Add metadata operation.
lpfnGetCB	LPAFT_IG_METAD_ITEM_GET_CB	New value of callback function of type LPAFT_IG_METAD_ITEM_GET_CB that is to be used for Get metadata operation.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
VOID ACCUAPI MetaDataGet(
        LPVOID   lpPrivate,             // Private callback data.
        AT_MODE  FilterID,
        LPCHAR   ItemName,             // Name of data item
        DWORD    ItemID,             // ID of data item
        AT_MODE  ItemType,             // Type of item
        LPVOID   ItemValue,             // value of item
        AT_MODE  ValueType,             // type of value
        DWORD    ValueLength,         // length of value
        AT_BOOL  ReadOnlyValue         // inform about is value is changeable or not
)
{
    // ...
}
BOOL ACCUAPI MetaDataSet(
   LPVOID         lpPrivate,        // Private callback data.
   AT_MODE        FilterID,
   LPCHAR         ItemName,            // Name of data item
   DWORD          ItemID,            // ID of data item
   AT_MODE        ItemType,            // Type of item
   LPVOID         ItemValue,        // value of item
   AT_MODE        ValueType,        // type of value
   DWORD          ValueLength,        // length of value
   AT_BOOL        ReadOnlyValue,    // inform about is value is changeable or not
   LPVOID         *NewItemValue,
   LPAT_MODE      NewValueType,
   LPDWORD        NewValueLength
)
{
    // ...
    return TRUE;
}
BOOL ACCUAPI MetaDataAdd(
        LPVOID  lpPrivate,            // Private callback data.
        AT_MODE FilterID,
        LPCHAR  *ItemName,            // Name of data item
        DWORD   *ItemID,            // ID of data item
        AT_MODE *ItemType,            // Type of item
        LPVOID  *ItemValue,            // value of item
        AT_MODE *ValueType,            // type of value
        DWORD   *ValueLength,        // length of value
        AT_BOOL *ReadOnlyValue        // inform about is value is changeable or not
)
{
    // ...
    return TRUE;
}

void Example_IG_fltr_metad_callback_set()
{
    AT_ERRCOUNT nErrcount;            // Count of returned errors on stack
    // Set metadata callback functions
    nErrcount = IG_fltr_metad_callback_set(NULL, MetaDataSet, MetaDataAdd, MetaDataGet);
}
```

Remarks:

See also the section Using Format Filters API for Filter Control.

1.3.1.2.10.21 IG_fltr_metad_update_file

This function creates a new file with an exact copy of the source file's pixel data and with new metadata.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_metad_update_file(
   const LPSTR lpszFileNameSrc,
   const LPSTR lpszFileNameDest,
   AT_LMODE lFormatType,
   UINT nPageNumber
);
```

Arguments:

Name	Type	Description
lpszFileNameSrc	const LPSTR	Path and name of the source file. The path can be absolute or relative.
lpszFileNameDest	const LPSTR	Path and name of the destination file. The path can be absolute or relative. Source and destination file names must be different.
lFormatType	AT_LMODE	Specifies the format of the source file. See enumIGFormats for possible values and also see Remarks.
nPageNumber	UINT	Page number for metadata updating.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

The function supports the following image file formats:

- TIFF (except TIFF-JPEG)
- JPEG

Example:

```
AT_ERRCOUNT     nErrcount;            // Count of returned errors on stack

nErrcount = IG_fltr_metad_update_file("picture.tif", "picture_new.tif", IG_FORMAT_TIF, 1);
```

Remarks:

Pixel data is not decoded but copied directly from source to destination file.

The function creates a new file that contains copy of source file data with the new metadata for required page. lFormatType parameter value should be the same as the source file format type. In a multipage file, the rest of pages are copied verbatim from source to destination file. The function obtains new metadata from metadata callback functions LPAFT_IG_METAD_ITEM_SET_CB and LPAFT_IG_METAD_ITEM_ADD_CB.

IG_fltr_metad_update_file() function can be used as follows:

- Use IG_fltr_pageinfo_get to get metadata for the page into your application's storage.
- Change metadata (add / delete / change metadata tags or metadata values).
- Call IG_fltr_metad_update_file() function, supplying the metadata tags to LPAFT_IG_METAD_ITEM_SET_CB and LPAFT_IG_METAD_ITEM_ADD_CB callbacks. nPageNumber and lFormatType parameter values should correspond to the loaded page and source file format.

Destination file will receive the copy of the source file, with new metadata for the specified page.

See Also:

Updating Non-Image Data without Loading and Saving the Image

1.3.1.2.10.22 IG_fltr_pagecount_FD_format

This function obtains the number of pages in the open multi-page file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_pagecount_FD_format(
   AT_MODE nFormat,
   AT_INT fd,
   LONG lOffset,
   LPUINT lpPageCount
);
```

Arguments:

Name	Type	Description
nFormat	AT_MODE	A constant indicating the file format of the input file. See enumIGFormats for possible values. Set to IG_FORMAT_UNKNOWN to let ImageGear detect the file format.
fd	AT_INT	Handle of the open file containing the image. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset to the image in the file.
lpPageCount	LPUINT	Return: Number of pages in an image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
UINT nPageCount;
AT_ERRCOUNT nErrCount;

HANDLE      fd;         //File Descriptor
fd = CreateFile(_T("picture.tif"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if(fd != INVALID_HANDLE_VALUE)
{
    nErrCount = IG_fltr_pagecount_FD_format(IG_FORMAT_TIF, (AT_INT)fd, 0, &nPageCount);
    CloseHandle(fd);
}
```

Remarks:

This function is similar to the IG_page_count_get_FD function, but has an additional parameter, nFormat, which specifies the file format of the input file.

If nFormat = IG_FORMAT_UNKNOWN then ImageGear attempts to detect the file format automatically, and then loads the image. Otherwise, ImageGear skips the file format detection and gets the page count using the specified format filter.

1.3.1.2.10.23 IG_fltr_pagecount_file_format

This function obtains the number of pages in a multi-page file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_pagecount_file_format(
   AT_MODE nFormat,
   const LPSTR lpszFileName,
   LPUINT lpPageCount
);
```

Arguments:

Name	Type	Description
nFormat	AT_MODE	A constant indicating the file format of the input file. See enumIGFormats for possible values. Set to IG_FORMAT_UNKNOWN to let ImageGear detect the file format.
lpszFileName	const LPSTR	Path and name of the file to get the page count for. The path can be absolute or relative.
lpPageCount	LPUINT	Number of pages returned by this function.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
UINT nPageCount;
AT_ERRCOUNT nErrCount = IG_fltr_pagecount_file_format(IG_FORMAT_TIF, "picture.tif",
&nPageCount);
```

Remarks:

If nFormat = IG_FORMAT_UNKNOWN then ImageGear attempts to detect the file format automatically, and then detects the page count. Otherwise, ImageGear skips the file format detection and counts image page using the specified format filter.

1.3.1.2.10.24 IG_fltr_pagedelete_file

This function deletes pages from a multipage file, if such operation is supported by the format filter.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_pagedelete_file(
   const LPSTR lpszFileName,
   AT_MODE nFormatType,
   UINT nStartPage,
   UINT nRange
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Path and name of the multipage file to delete pages from. The path can be absolute or relative.
nFormatType	AT_MODE	A constant indicating the file format of the input file. See enumIGFormats for possible values. Set to IG_FORMAT_UNKNOWN to let ImageGear detect the file format.
nStartPage	UINT	Determines the first page to delete from the szFileName file.
nRange	UINT	Determines the number of pages to delete, starting from nStartPage.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT     nErrcount;             // Count of returned errors on stack
nErrcount = IG_fltr_pagedelete_file("picture_multipage.tif", IG_FORMAT_TIF, 1, 1);
```

Remarks:

Use IG_fltr_info_get function to determine whether the format filter supports page deletion. If the flags returned by this function contain IG_FLTR_PAGEDELETESUPPORT, then the format filter supports the deleting procedure.

See also the section Getting Information and Sorting Images.

1.3.1.2.10.25 IG_fltr_pageinfo_get

This function obtains information about a page of a multipage file, without loading its pixel data. This is an obsolete function, see remarks.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_pageinfo_get(
   const LPSTR lpszFileName,
   UINT nPage,
   LPAT_MODE lpFileType,
   LPAT_MODE lpCompression,
   LPAT_DIB lpDIB
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Path and name of the file. The path can be absolute or relative.
nPage	UINT	Number of the page in a multi-page file for which the information should be obtained.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values.
lpCompression	LPAT_MODE	Pointer to an AT_MODE variable in which compression type will be returned. See enumIGCompressions for possible values.
lpDIB	LPAT_DIB	Pointer to an AT_DIB structure to which other file information, such as width, height, and Bits Per Pixel will be returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

Example:

```
AT_ERRCOUNT nErrCount;
AT_MODE fileType;
AT_MODE compression;
AT_DIB atDib;

nErrCount = IG_fltr_pageinfo_get("picture.tif", 1,
    &fileType, &compression, &atDib);
```

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_fltr_pageinfo_get_ex instead.

See also the section Getting Information and Sorting Images.

1.3.1.2.10.26 IG_fltr_pageinfo_get_ex

This function obtains information about the page specified by the nPage parameter from a named multipage file without actually loading it.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_pageinfo_get_ex(
   const LPSTR lpszFileName,
   UINT nPage,
   LPAT_MODE lpFileType,
   LPAT_MODE lpCompression,
   HIGDIBINFO* lphDIB
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Path and name of the file to get the information about. The path can be absolute or relative.
nPage	UINT	Number of the page in a multi-page file for which to get information.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values.
lpCompression	LPAT_MODE	Pointer to an AT_MODE variable in which compression type will be returned. See enumIGCompressions for possible values.
lphDIB	HIGDIBINFO*	Pointer to HIGDIBINFO object to which other file information, such as width, height, Bits Per Pixel etc. will be returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrCount;
AT_MODE fileType;
AT_MODE compression;
HIGDIBINFO hDIB;

nErrCount = IG_fltr_pageinfo_get_ex("picture.tif", 1,
    &fileType, &compression, &hDIB);
if(nErrCount == 0)
{
    // ...
    // Delete DIB info
    IG_DIB_info_delete(hDIB);
}
```

Remarks:

Any of the output parameters such as lpFileType, lpCompression or lphDIB can be NULL, if the corresponding info is not required.

See also the section Getting Information and Sorting Images.

This function is identical to IG_info_get_ex.

1.3.1.2.10.27 IG_fltr_pageswap_file

This function swaps two pages in a multipage file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_pageswap_file(
   const LPSTR lpszFileName,
   AT_MODE nFormatType,
   UINT nPage1,
   UINT nPage2
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Path and name of the file to swap the pages in. The path can be absolute or relative.
nFormatType	AT_MODE	A constant indicating the file format of the input file. See enumIGFormats for possible values. Set to IG_FORMAT_UNKNOWN to let ImageGear detect the file format.
nPage1	UINT	Number of page 1 to swap with page 2.
nPage2	UINT	Number of page 2 to swap with page 1.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT     nErrcount;             // Count of returned errors on stack
nErrcount = IG_fltr_pageswap_file("picture_multipage.tif", IG_FORMAT_TIF, 1, 2);
```

Remarks:

Use IG_fltr_info_get function to determine whether the format filter supports page swapping. If the flags returned by this function contain IG_FLTR_PAGESWAPSUPPORT, then the format filter supports the swapping procedure.

See also the section Getting Information and Sorting Images.

1.3.1.2.10.28 IG_fltr_raster_plane_callback_get

This function allows you to retrieve the settings of raster plane callback LPFNIG_RASTER_PLANE_SET function.

Declaration:

```
AT_ERRCOUNT LACCUAPI IG_fltr_raster_plane_callback_get(
   LPFNIG_RASTER_PLANE_SET* lplpfnRasterPlaneSetCB,
   AT_VOID** lpReserved
);
```

Arguments:

Name	Type	Description
lplpfnRasterPlaneSetCB	LPFNIG_RASTER_PLANE_SET*	SET callback function.
lpReserved	AT_VOID**	Reserved for Get function.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.///

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;                      // Count of returned errors on stack
LPFNIG_RASTER_PLANE_SET lpfnRasterPlaneCB;    // Raster Plane callback function

// Get ICC callback functions
nErrcount = IG_fltr_raster_plane_callback_get(&lpfnRasterPlaneCB, NULL);
```

Remarks:

ImageGear calls lpfnRasterPlaneSetCB callback function to pass raster plane data that has been read from a file to the application. The callback is invoked when reading images where pixel data is stored in planar format. As of this writing, only TIF and DICOM format filters support this callback.

1.3.1.2.10.29 IG_fltr_raster_plane_callback_set

This function allows you to register your raster plane callback LPFNIG_RASTER_PLANE_SET function.

Declaration:

```
AT_ERRCOUNT LACCUAPI IG_fltr_raster_plane_callback_set(
   LPFNIG_RASTER_PLANE_SET lpfnRasterPlaneSetCB,
   AT_VOID* Reserved
);
```

Arguments:

Name	Type	Description
lpfnRasterPlaneSetCB	LPFNIG_RASTER_PLANE_SET	Specifies the callback function which will receive raster planes of pixels.
Reserved	AT_VOID*	Reserved, should be NULL.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
// Handles merging of planes into a raster line
AT_ERRCOUNT ACCUAPI RasterPlaneSet(
    AT_VOID                *lpPrivate,        // Private data passed in
    const AT_VOID*      lpRast,                // Raster line to set
    AT_PIXPOS             cyPos,             // Y position in the image
    AT_INT                cRasterSize,        // Size of the raster line
    AT_INT                 nBitPlane         // Bit plane to merge in
    )
{
    // ...
    return 0;
}

void Example_IG_fltr_raster_plane_callback_set()
{
    AT_ERRCOUNT nErrcount;            // Count of returned errors on stack
    // Set raster plane callback functions
    nErrcount = IG_fltr_raster_plane_callback_set(RasterPlaneSet, NULL);
}
```

Remarks:

ImageGear calls lpfnRasterPlaneSetCB callback function to pass raster plane data that has been read from a file to the application. The callback is invoked when reading images where pixel data is stored in planar format. As of this writing, only TIF and DICOM format filters support this callback.

1.3.1.2.10.30 IG_fltr_save_FD_size_calc

This function is used to determine the size that is required for saving an image to a file in the specified format.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_save_FD_size_calc(
   HIGEAR hIGear,
   AT_INT reserved_fd,
   AT_LMODE lFormatType,
   AT_UINT reserved_page,
   AT_BOOL reserved_overwrite,
   LPAT_UINT lpFileSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image on which to calculate the size.
reserved_fd	AT_INT	Reserved for future use. Set to 0.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats for possible values.
reserved_page	AT_UINT	Reserved for future use. Set to 0.
reserved_overwrite	AT_BOOL	Reserved for future use. Set to FALSE.
lpFileSize	LPAT_UINT	Returns the maximum possible size of the saved file.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrCount;

HIGEAR hIGear = 0;
AT_UINT nFileSize;    // File size returned;
nErrCount = IG_load_file("picture.tif", &hIGear);
if(nErrCount == 0)
{
    nErrCount = IG_fltr_save_FD_size_calc(hIGear, 0, IG_SAVE_TIF_UNCOMP, 1, TRUE,
&nFileSize);
    IG_image_delete(hIGear);
}
```

Remarks:

This call may be used prior to calling IG_save_FD to determine the size of result file.

As of this writing, the function can only calculate the size of a single-page file. To calculate the size of a multipage file after addition of a page, load the original file into a memory buffer, and then use IG_fltr_save_mem_size_calc.

See also the section Using Format Filters API for Image Saving.

1.3.1.2.10.31 IG_fltr_save_file

This function stores the image referenced by hIGear to a file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_save_file(
   HIGEAR hIGear,
   const LPSTR lpszFileName,
   AT_LMODE lFormatType,
   UINT nPageNumber,
   AT_BOOL bOverwrite
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image to save.
lpszFileName	const LPSTR	Path and name of the file to save the image to. The path can be absolute or relative.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats for possible values.
nPageNumber	UINT	Specifies the page number of the page inserted into a multi-page file. Note that page numbers begin at 1, not 0. Set to 0 to append the page after the last page of the source file. Set to 1 if the file format does not support multipage, or if saving to a new file.
bOverwrite	AT_BOOL	Set to TRUE to overwrite existing file during the saving. Set to FALSE to insert or append the page to the file, if the format supports multipage saving.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT     nErrcount;             // Count of returned errors on stack
HIGEAR hIGear;                         //ImageGear handle
nErrcount = IG_load_file("picture.tif", &hIGear);
if(nErrcount == 0)
{
    // Save image to file "picture.bmp" in BMP format without compression:
    nErrcount = IG_fltr_save_file(hIGear, "picture_new.tif", IG_SAVE_TIF_UNCOMP, 1, TRUE);
    IG_image_delete(hIGear);
}
```

Remarks:

lFormatType is used to set the format and compression (if applicable) of the output file. If you want to have ImageGear use the file extension provided in your filename string (lpszFilename) to determine the file format in which to save the file, set lFormatType = IG_SAVE_UNKNOWN.

See also the section Using Format Filters API for Image Saving.

1.3.1.2.10.32 IG_fltr_save_file_size_calc

This function is used to determine the size that is required for saving the image to the file in the given format.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_save_file_size_calc(
   HIGEAR hIGear,
   const LPSTR reserved_filename,
   AT_LMODE lFormatType,
   AT_UINT reserved_page,
   AT_BOOL reserved_overwrite,
   LPAT_UINT lpFileSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image for which to calculate the size.
reserved_filename	const LPSTR	Reserved for future use. Set to NULL.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats for possible values.
reserved_page	AT_UINT	Reserved for future use. Set to 0.
reserved_overwrite	AT_BOOL	Reserved for future use. Set to FALSE.
lpFileSize	LPAT_UINT	Returns the maximum possible size of the file.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrCount;

HIGEAR hIGear = 0;
AT_UINT nFileSize;    // File size returned;
nErrCount = IG_load_file("picture.tif", &hIGear);
if(nErrCount == 0)
{
    nErrCount = IG_fltr_save_file_size_calc(hIGear, 0, IG_SAVE_TIF_UNCOMP, 1, TRUE,
&nFileSize);
    IG_image_delete(hIGear);
}
```

Remarks:

This call may be used prior to calling IG_fltr_save_file to determine the size of result file.

As of this writing, the function can only calculate the size of a single-page file. To calculate the size of a multipage file after addition of a page, load the original file into a memory buffer, and then use IG_fltr_save_mem_size_calc.

See also the section Using Format Filters API for Image Saving.

1.3.1.2.10.33 IG_fltr_save_mem

This function stores the image referenced by hIGear to the specified memory buffer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_save_mem(
   HIGEAR hIGear,
   LPVOID lpImage,
   AT_UINT nImageSize,
   AT_UINT nBufferSize,
   AT_LMODE lFormatType,
   AT_UINT nPageNumber,
   AT_BOOL bOverwrite,
   LPAT_UINT lpActualSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image to save.
lpImage	LPVOID	Pointer to first byte of memory area in which to save.
nImageSize	AT_UINT	Size of image (if exists).
nBufferSize	AT_UINT	Size of memory block.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats for possible values.
nPageNumber	AT_UINT	Specifies the page number of the page inserted into a multi-page file. Note that page numbers begin at 1, not 0. Set to 0 to append the page after the last page of the source file. Set to 1 if the file format does not support multipage, or if saving to a new file.
bOverwrite	AT_BOOL	Set to TRUE to overwrite existing file during the saving. Set to FALSE to insert or append the page to the file, if the format supports multipage saving.
lpActualSize	LPAT_UINT	Size of new or updated file in memory.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrCount;

HIGEAR hIGear = 0;
AT_UINT nFileSize;    // File size returned;
nErrCount = IG_load_file("picture.tif", &hIGear);
if(nErrCount == 0)
{
    // Get required memory size
    nErrCount = IG_fltr_save_mem_size_calc(hIGear, NULL, 0, IG_SAVE_TIF_UNCOMP, 1, TRUE,
&nFileSize);
    if(nErrCount == 0)
    {
        // Allocate memory
        LPAT_BYTE memBuffer = (LPAT_BYTE)malloc(nFileSize);
        nErrCount = IG_fltr_save_mem(hIGear, memBuffer, 0, nFileSize, IG_SAVE_TIF_UNCOMP,
1, TRUE, &nFileSize);

        //...

        free(memBuffer);
    }
    IG_image_delete(hIGear);
}
```

Remarks:

lFormatType is used to set the format and compression (if applicable) of the output file. If you want to have ImageGear use the file extension provided in your filename string (lpszFilename) to determine the file format in which to save the file, set lFormatType = IG_SAVE_UNKNOWN.

Before using this function, the application must allocate a memory buffer, sufficient for storing the saved image. Use IG_fltr_save_mem_size_calc to determine the necessary buffer size.

This function is similar to the IG_save_mem function, but it allows you to insert a new page in multi-page file as either the end page or the page with a given nPageNumber number.

See also the section Using Format Filters API for Image Saving.

1.3.1.2.10.34 IG_fltr_save_mem_size_calc

This function is used to determine the size that is required for saving the image to the file or memory buffer in the given format.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_save_mem_size_calc(
   HIGEAR hIGear,
   LPVOID lpImage,
   AT_UINT nImageSize,
   AT_LMODE lFormatType,
   AT_UINT nPageNumber,
   AT_BOOL reserved_overwrite,
   LPAT_UINT lpFileSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image on which to calculate the size.
lpImage	LPVOID	If the buffer exists and already contains an image file to which a page will be appended, this parameter specifies a pointer to first byte of the existing file in the memory buffer.
nImageSize	AT_UINT	If the buffer exists and already contains an image file to which a page will be appended, this parameter specifies the size of existing image.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats for possible values.
nPageNumber	AT_UINT	Specifies the page number of the page inserted into a multi-page file. Note that page numbers begin at 1, not 0. Set to 0 to append the page after the last page of the source file. Set to 1 if the file format does not support multipage, or if saving to a new file.
reserved_overwrite	AT_BOOL	Reserved for future use. Set to FALSE.
lpFileSize	LPAT_UINT	Returns the maximum possible size of the file.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrCount;

HIGEAR hIGear = 0;
AT_UINT nFileSize;    // File size returned;
nErrCount = IG_load_file("picture.tif", &hIGear);
if(nErrCount == 0)
{
    nErrCount = IG_fltr_save_mem_size_calc(hIGear, NULL, 0, IG_SAVE_TIF_UNCOMP, 1, TRUE,
&nFileSize);
    IG_image_delete(hIGear);
}
```

Remarks:

This function can be used prior to calling IG_fltr_save_mem to determine the amount of memory that needs to be allocated.

This function supports the calculation of a multipage image file size after addition of a page. If a file exists in the memory buffer before calling this function, and the file format supports appending pages, the function calculates the size of the file after appending the page.

See also the section Using Format Filters API for Image Saving.

1.3.1.2.10.35 IG_fltr_savelist_get

This function prepares the list of constants corresponding to format and compression combinations available for saving of the specified image. This is an obsolete function, see remarks.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_savelist_get(
   LPAT_DIB lpDIB,
   LPAT_MODE lpnFilterList,
   UINT nFListSize,
   LPAT_LMODE lpSaveList,
   UINT nSListSize,
   LPUINT lpnSListCount
);
```

Arguments:

Name	Type	Description
lpDIB	LPAT_DIB	Pointer to the AT_DIB structure that contains the image parameters. If the value is not NULL, this function returns the list of enumIGSaveFormats values corresponding to saving formats (format and compression combinations) available for saving of the specified image. If the value is NULL, then the function returns the list of all currently supported saving formats for file formats specified by lpnFilterList. If both the lpDIB and lpnFilterList are null, the function returns the list of all currently supported saving formats.
lpnFilterList	LPAT_MODE	Pointer to the list of format identifiers, which will be used in the save list. See enumIGFormats for possible values. If this parameter is NULL, then all currently supported formats will be used.
nFListSize	UINT	Array containing the number of elements if lpnFilterList is not NULL.
lpSaveList	LPAT_LMODE	Array containing the returned saving format constants. You can set this value to NULL if you only need to obtain the total number of found saving formats.
nSListSize	UINT	Size of the lpSaveList array.
lpnSListCount	LPUINT	If the lpSaveList array is not NULL, this parameter returns the number of copied enumIGSaveFormats values. If lpSaveList is NULL, this parameter returns the total number of records.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

Example:

```
AT_ERRCOUNT nErrCount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
UINT nCount;            // Number of save formats
LPAT_LMODE lpSaveList;

// Load the image
nErrCount = IG_load_file("picture.tif", &hIGear);
if( nErrCount == 0 )
```

```
{
    AT_DIB atDib;
    AT_DIMENSION nWidth, nHeight;
    UINT nBitsPerPixel;
    // Get image info
    nErrCount = IG_image_dimensions_get(hIGear, &nWidth, &nHeight, &nBitsPerPixel);
    // Fill in AT_DIB structure
    memset(&atDib, 0, sizeof(AT_DIB));
    atDib.biSize = sizeof(AT_DIB);
    atDib.biWidth = nWidth;
    atDib.biHeight = nHeight;
    atDib.biPlanes = 1;
    atDib.biBitCount = nBitsPerPixel;

    // Get save formats count
    nErrCount = IG_fltr_savelist_get(&atDib, NULL, 0, NULL, 0, &nCount);
    // Allocate memory
    lpSaveList = (LPAT_LMODE)malloc( nCount*sizeof(AT_LMODE) );
    if( lpSaveList!=NULL )
    {
        // Get save list
        nErrCount = IG_fltr_savelist_get(&atDib, NULL, 0, lpSaveList, nCount, NULL);

        //...

        // Delete memory
        free(lpSaveList);
    }
    // Delete the image
    IG_image_delete(hIGear);
}
```

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_fltr_savelist_get_ex instead.

Records returned by the function are sorted alphabetically by their short names. Short names correspond to those returned by IG_fltr_info_get function.

This function works similarly to IG_fltr_compressionlist_get, but it works with all formats supported by ImageGear rather than with a particular format. Values returned in the lpSaveList can be passed directly to ImageGear saving functions such as IG_fltr_save_file.

See also the section Using Format Filters API for Image Saving.

1.3.1.2.10.36 IG_fltr_savelist_get_ex

This function prepares the list of constants corresponding to format and compression combinations available for saving of the specified image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_fltr_savelist_get_ex(
   const HIGDIBINFO hDIB,
   LPAT_MODE lpnFilterList,
   UINT nFListSize,
   LPAT_LMODE lpSaveList,
   UINT nSListSize,
   LPUINT lpnSListCount
);
```

Arguments:

Name	Type	Description
hDIB	const HIGDIBINFO	Handle of DIB info object that contains image parameters. If the value is not NULL, this function returns the list of enumIGSaveFormats values corresponding to saving formats (format and compression combinations) available for saving of the specified image. If the value is NULL, then the function returns the list of all currently supported saving formats for file formats specified by lpnFilterList. If both the hDIB and lpnFilterList are null, the function returns the list of all currently supported saving formats.
lpnFilterList	LPAT_MODE	Pointer to the list of format identifiers, which will be used in the save list. See enumIGFormats for possible values. If this parameter is NULL, then all currently supported formats will be used.
nFListSize	UINT	Array containing the number of elements if lpnFilterList is not NULL.
lpSaveList	LPAT_LMODE	Array containing the returned saving format constants. You can set this value to NULL if you only need to obtain the total number of found saving formats.
nSListSize	UINT	Size of the lpSaveList array.
lpnSListCount	LPUINT	If the lpSaveList array is not NULL, this parameter returns the number of copied enumIGSaveFormats values. If lpSaveList is NULL, this parameter returns the total number of records.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrCount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
UINT nCount;            // Number of save formats
HIGDIBINFO hDIB;        // DIB info handle of image
LPAT_LMODE lpSaveList;

// Load the image
nErrCount = IG_load_file("picture.tif", &hIGear);
if( nErrCount == 0 )
{
    // Get DIB info
    nErrCount = IG_image_DIB_info_get(hIGear, &hDIB);
    // Get save formats count
    nErrCount = IG_fltr_savelist_get_ex(hDIB, NULL, 0, NULL, 0, &nCount);
    // Allocate memory
    lpSaveList = (LPAT_LMODE)malloc( nCount*sizeof(AT_LMODE) );
    if( lpSaveList!=NULL )
    {
        // Get save list
        nErrCount = IG_fltr_savelist_get_ex(hDIB, NULL, 0, lpSaveList, nCount, NULL);

        //...

        // Delete memory
        free(lpSaveList);
        // Delete DIB info
        IG_DIB_info_delete(hDIB);
    }
    // Delete the image
    IG_image_delete(hIGear);
}
```

Remarks:

Records returned by the function are sorted alphabetically by their short names. Short names correspond to those returned by IG_fltr_info_get function.

This function works similarly to IG_fltr_compressionlist_get_ex, but it works with all formats supported by ImageGear rather than with a particular format. Values returned in the lpSaveList can be passed directly to ImageGear saving functions such as IG_fltr_save_file.

See also the section Using Format Filters API for Image Saving.

1.3.1.2.11 FX Functions

This section provides information about the FX group of functions.

- IG_FX_blur
- IG_FX_chroma_key
- IG_FX_diffuse
- IG_FX_emboss
- IG_FX_motion
- IG_FX_noise
- IG_FX_pixelate
- IG_FX_posterize
- IG_FX_spotlight
- IG_FX_stitch
- IG_FX_texture
- IG_FX_twist
- IG_FX_watermark

1.3.1.2.11.1 IG_FX_blur

This function blurs an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_blur (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const AT_MODE nBlurMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nBlurMode	const AT_MODE	A constant such as IG_BLUR_3 specifying the kernel size to use to perform the blurring. See file accucnst.h for the IG_BLUR_ constants available.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette

Example:

```
HIGEAR    hIGear;         /* HIGEAR handle of image    */
AT_RECT   rcImageRect;    /* Image's current image rectangle */
/* Blur only the image rect portion, using a 5 x 5 kernel:     */
IG_FX_blur ( hIGear, &rcImageRect, IG_BLUR_5 );
```

Remarks:

nBlurMode controls the degree of blurring.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.2 IG_FX_chroma_key

This function blends two images, inserting the pixel values from hIGearBkGrnd wherever the pixel in hIGearFrGrnd is in the specified hue range.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_chroma_key (
        HIGEAR hIGearFrGrnd,
        LPAT_RECT lpRect,
        HIGEAR hIGearBkGrnd,
        const DOUBLE dblHueCenter,
        const DOUBLE dblHueRange,
        const UINT nSmooth,
        const UINT nThreshold
);
```

Arguments:

Name	Type	Description
hIGearFrGrnd	HIGEAR	HIGEAR handle of image to modify where the specified hue is found.
lpRect	LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image to be processed. Use NULL for the whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
hIGearBkGrnd	HIGEAR	HIGEAR handle of an image to insert from on hue match.
dblHueCenter	const DOUBLE	The angle in degrees (in the standard Color Wheel) of the hue to match. 0.0 - 360.0 (360 == 0).
dblHueRange	const DOUBLE	The range on which to allow (in degrees) either side. 0.0 - 360.0.
nSmooth	const UINT	An integer from 0 to 25 specifying how much to smooth the transition. 0 gives the sharpest edge.
nThreshold	const UINT	Intensity below which to ignore the hue, and fail the match (0 - 255).

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB – 1 bpp
Grayscale – 1 bpp

Example:

```
HIGEAR hiGear, /* HIGEAR handle of image to blend into  */
hIGear Bkgrnd; /* HIGEAR handle of image to blend in  */
AT_PIXEL pixelvalue[3]; /* 3 bytes for return of an RGB pixel value */
DOUBLE Hue;     /* Hue angle that will be returned */
/* Retrieve the RGB value of the pixel at (10,20) in the HIGEAR image */
nErrcount = IG_DIB_pixel_get (hIGear, 10, 20, &pixelvalue[0]);
/* Pass the RGB pixel value to IG_IP_RGB_to_hue to convert to hue angle  */
nErrcount = IG_IP_RGB_to_hue (&pixelvalue[0], &Hue);
/* Pass newly calculated hue angle to chroma_key to combine images   */
IG_FX_chroma_key ( hIGear, NULL, hIGearBkgrnd, Hue, 10.0, 0, 20 );
```

Remarks:

You can control the smoothness of the transitions using argument nSmooth, and you can prevent the hue of dark pixels from being considered, using nThreshold. To determine the proper hue center and hue range, you may want to use the IG_DIB_pixel_get() and IG_IP_RGB_to_hue() functions.

See also IG_IP_blend_with_LUT(), IG_DIB_pixel_get() and IG_IP_RGB_to_hue() functions.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

See the descriptions of IG_IP_NR_ROI_mask_associate and IG_IP_NR_ROI_to_HIGEAR_mask for more details. The "background image" must have the same height, width, and bit depth as the "foreground image."

The hue is not an 8-bit HSI. HSI is the name of the color space.

1.3.1.2.11.3 IG_FX_diffuse

This function diffuses an image by shuffling the positions of pixels.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_diffuse (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const UINT nStrength
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nStrength	const UINT	An integer from 1 to 16 specifying the amount of diffusion.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette

Example:

```
HIGEAR    hIGear;        /* HIGEAR handle of image */
/* Diffuse the image slightly: */
IG_FX_diffuse ( hIGear, NULL, 3 );
```

Remarks:

The greater the value of nStrength, the greater the diffusion.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.4 IG_FX_emboss

This function produces an embossed or 3-D like chiseled-in-stone look to the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_emboss (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const DOUBLE dblStrength,
        const AT_MODE nCompassDir
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
dblStrength	const DOUBLE	The embossing strength. The valid range is from 1.0 to 5.0.
ncompassDir	const AT_MODE	An AT_MODE Compass direction constants (see accucnst.h file): • IG_COMPASS_N - North direction • IG_COMPASS_NE - North-East direction • IG_COMPASS_E - East direction • IG_COMPASS_SE - South-East direction • IG_COMPASS_S - South direction • IG_COMPASS_SW - South-West direction • IG_COMPASS_W - West direction • IG_COMPASS_NW - North-West direction

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette;
Images that have a Grayscale LUT attached to them.

Example:

```
HIGEAR hIGear;            /* HIGEAR handle of image  */
AT_RECT rcImageRect;     /* Image's current image rectangle  */
/* Emboss only the image rect portion: */
IG_FX_emboss ( hIGear, &rcImageRect, 3.0, IG_COMPASS_NE );
```

Remarks:

The result looks similar to the engraved face of a coin. The greater the value of dblStrength, the greater the effect will be (higher ridges and lower depressions). The direction in which the image will appear elevated is selected by nCompassDir.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.5 IG_FX_motion

This function makes the image look as though it was moving when the image was captured.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_motion (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const UINT nExtent,
        const AT_MODE nDirection
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for the whole image to be processed. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nExtent	const UINT	Set to a UINT for the amount of motion you would like applied. If nDirection will be set to "S, W, E, N", the correct range for this variable is 2 - 15. If nDirection will be set to "SE, NE, NW, SW", the correct range for this variable is 3 - 22. This variable determines the extent to which the pixels will be "moved" or "smeared", or literally how many pixel lengths each pixel will "move over."
nDirection	const AT_MODE	An AT_MODE Compass direction constant. Please see accucnst.h file or the description of IG_FX_emboss() function for full list of these constants.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of image */
/* Blur whole image to imply fast motion to the south-east:  */
IG_FX_motion ( hIGear, NULL, 6, IG_COMPASS_SE );
```

Remarks:

The amount of motion depends on the nAmount parameter. A larger nAmount makes the motion appear faster. Use nDirection to select which direction the image appears to be moving toward. lpRect specifies what portion of the image is to be affected.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.6 IG_FX_noise

This function is used to create noise in an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_noise (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const WORD nType,
        const DOUBLE dblStrength,
        const INT nHitRate,
        const DOUBLE dblSigma,
        const AT_MODE nColorChannel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nType	const WORD	An IG_NOISE constant such as IG_NOISE_LINEAR, IG_NOISE_GAUSSIAN.
dblStrength	const DOUBLE	From 0.0 to 127.0, specifying the degree of noise alteration to introduce into affected pixels.
nHitRate	const INT	Set = 1 to add noise to all pixels, > 1 to skip pixels, only adding noise to some. Larger values cause fewer pixels to be affected. Valid range: 1 to 500.
dblSigma	const DOUBLE	Used with IG_NOISE_GAUSSIAN, range: 0.1-25.0.
nColorChannel	const AT_MODE	An IG_COLOR_COMP_ constant. See file accucnst.h for the full list of these constants.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR    hIGear;  /* HIGEAR handle of image   */
/* Add moderate noise to about 5 percent of the pixels: */
IG_FX_noise ( hIGear, NULL, IG_NOISE_LINEAR, 30.0, 20, 1.0, IG_COLOR_COMP_RGB );
```

Remarks:

This effect can make an image look older. nHitRate selects how many pixels may have noise introduced into them. dblStrength determines how strongly a pixel's value is to be altered when it is selected to be altered. nType determines the algorithm used to determine the noise alteration.

nHitRate = 1 indicates that almost every pixel should be affected. A value of 100 would indicate that every 100th pixel should be affected. dblStrength = 1.0 indicates that a value of about +1 to -1 should be added to those pixels selected while dblStrength = 50.0 would increase the noise result by adding values in the range of -50 to +50.

Which pixels are altered and the amount of noise to apply are selected randomly.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.7 IG_FX_pixelate

This function redraws an image using what appear to be very large pixels.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_pixelate (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const AT_DIMENSION nXRes,
        const AT_DIMENSION nYRes,
        const AT_MODE nResampleIn,
        const AT_MODE nResampleOut,
        const WORD radius,
        const LPAT_RGB lprgbBkColor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nXRes	const AT_DIMENSION	The distance between the new pixels in the horizontal direction.
nYRes	const AT_DIMENSION	The distance between the new pixels in the vertical direction.
nResampleIn	const AT_MODE	An IG_RESAMPLE_IN_ constant. These are listed in accucnst.h
nResampleOut	const AT_MODE	IG_RESAMPLE_OUT_SQUARE or _CIRCLE, the type of result to produce.
nRadius	const WORD	Radius of new "pixels", in pixels. Only applicable nResampleOut=IG_RESAMPLE_OUT_CIRCLE.
LprgbBkColor	const LPAT_RGB	If circle, a far pointer to an AT_RGB struct specifying the background color surrounding the output circles.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image */
AT_ERRCOUNT nErrcount;
nErrcount = IG_FX_pixelate ( hIGear, NULL, 10, 10,
IG_RESAMPLE_IN_AVE,IG_RESAMPLE_OUT_SQUARE, 0, NULL );
```

Remarks:

Use nXRes and nYResto specify how many "apparent pixels" you want in the result.

nResampleIn tells ImageGear what to do with the block of pixels it reads in. It lets you have each output "apparent pixel" computed on the basis of the average, minimum, maximum, or central pixel in the block. Setting this to IG_RESAMPLE_IN_AV will set the value of all pixels in the block to the average all of the pixel values in the block.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.8 IG_FX_posterize

Posterize reduces the number of actual colors in the image by creating a "stair case" in the palette.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_posterize (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const WORD nLevels,
        const AT_MODE nColorChannel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying a rectangular portion of the image to be processed. NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nLevels	const WORD	Number of unique colors or levels wanted in the resulting image. Valid range: 1-255.
nColorChannel	const AT_MODE	IG_COLOR_COMP_RGB, or use IG_COLOR_COMP_R, _B, or _G to affect only one color channel.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette.
Images that have a Grayscale LUT attached to them.

The function does not have any effect on 1 bpp images.

Example:

```
HIGEAR hIGear;        /* HIGEAR handle of image  */
/* Use only 50 colors, regardless of how many unique pixel values:  */
IG_FX_posterize ( hIGear, NULL, 50, IG_COLOR_COMP_RGB );
```

Remarks:

The number of steps wanted is specified by nLevels, in the range 1 to 255. nLevels = 255 would cause no effect while nLevels = 20 would cause 20 equally spaced steps to be created.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.9 IG_FX_spotlight

This function produces a "spotlight" effect within the circle specified by nRadius and (nCenterX, nCenterY).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_spotlight (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const AT_PIXPOS nCenterX,
        const AT_PIXPOS nCenterY,
        const AT_DIMENSION nRadius,
        const UINT nDarkenBy,
        const AT_PIXEL nSmoothing
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of an image into which to place the spotlight effect.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying a rectangular portion of the image to be processed. NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nCenterX	const AT_PIXPOS	X coordinate of the center of the circle to receive a spotlight effect; 0 to Width - 1.
nCenterY	const AT_PIXPOS	Y coordinate of the center of the circle; 0 to Height - 1.
nRadius	const AT_DIMENSION	Radius of the circle, in pixels; 2 to Height - 2.
nDarkenBy	const UINT	How much to darken the remainder of the image.
nSmoothing	const AT_PIXEL	0 to 255, specifying how much smoothing to apply at the edges; 0 = sharp edges, > 0 = smoother edges.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette;
Images that have a Grayscale LUT attached to them.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of image   */
AT_PIXPOS nXc, nYc;  /* Coords of center of spotlight area */
AT_DIMENSION nWid, nHi; /* Will receive width and height of image */
UINT nBpp;     /* Bits per pixel, not used */
AT_DIMENSION nRadius;  /* Radius of spotlight  */
IG_image_dimensions_get ( hIGear, &nWid, &nHi, &nBpp ); /* Get Wid,Hi */
nXc = nWid / 2;   nYc = nHi / 2;   /* Coords of center */
/* Diameter will be half of smallest dimension:   */
nRadius =  (nWid < nHi) ?  nWid / 4   :  nHi / 4;
/* Darken outside the circle by 40, with some smoothing of transition:  */
```

```
IG_FX_spotlight ( hIGear, NULL, nXc, nYc, nRadius, 40, 10 );
```

Remarks:

This function leaves the pixels within the circle unchanged while darkening the surrounding pixels by reducing their intensity. Use nDarkenBy to specify the reduction in intensity of the surrounding pixels.

Use nSmoothing to specify the amount of smoothing at the perimeter of the circle. nSmoothing = 0 will leave the sharpest edge.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. (See above.) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.10 IG_FX_stitch

This function produces an effect similar to IG_FX_emboss(), except that the output more closely resembles a quilted stitch pattern.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_stitch (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        AT_MODE nCompassDir,
        const DOUBLE dblStrength,
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying a rectangular portion of the image to be processed. NULL for the whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nCompassDir	AT_MODE	An AT_MODE Compass direction constant. Please see accucnst.h file or the description of IG_FX_emboss() function for full list of these constants.
dblStrength	const DOUBLE	0.0 to 5.0 (A very low value will produce a gray screen with no detail.)

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB – 1 bpp;
Grayscale – 1 bpp.

Example:

```
HIGEAR hIGear;          /* HIGEAR handle of image */
AT_RECT rcImageRect;    /* Image's current image rectangle */
/* Emboss only the image rect portion: */
IG_FX_stitch ( hIGear, &rcImageRect, IG_COMPASS_NE, 3.0 );
```

Remarks:

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. (See above.) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.11 IG_FX_texture

This function applies a texture to an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_texture (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const HIGEAR hTextureImage
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to which to apply texture.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non rectangular ROI defined by the mask.
hTextureImage	const HIGEAR	HIGEAR handle of the 8 bit image to be applied to image hIGear to produce textured effect.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;    /* Handle of image to be textured */
HIGEAR hTextureImage; /* Handle of 8 x 8 pixel 8-bit gray level texturing image */
/* Apply texture to the whole image: */
IG_FX_texture ( hIGear, NULL, hTextureImage );
```

Remarks:

The texturing image is a small 8-bit grayscale image that is treated as a sign centered image. Pixels in the texture image that are 127 have no effect on the original image, and the farther a texture image pixel is from 127, the greater its effect. The texture image is tiled over the entire original image starting in the top left corner. Any left over is clipped.

Sign centered images can be created using the emboss function.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

See the descriptions of IG_IP_NR_ROI_mask_associate and IG_IP_NR_ROI_to_HIGEAR_mask for more details.

1.3.1.2.11.12 IG_FX_twist

This function applies a special effect that makes the image look as if it is being viewed through a shower curtain.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_twist (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const AT_MODE nTwistType,
        const UINT nSquareSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image to be processed. Use NULL for whole image. Before ImageGear performs this operation, it determines if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear overrides the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nTwistType	const AT_MODE	One of the IG_TWIST_ constants: IG_TWIST_90IG_TWIST_180IG_TWIST_270IG_TWIST_RANDOM.
nSquareSize	const UINT	Size in pixels of the regions to which to apply twisting (valid range: 2 to 50).

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;     /* Handle of image to apply twisting to  */
/* Apply random twisting to 16 x 16 pixel squares of image:  */
IG_FX_twist ( hIGear, NULL, IG_TWIST_RANDOM, 16 );
```

Remarks:

The image can still be seen but it is chopped up so that detail is lost.

Each square of pixels of size nSquareSize in the image is rotated according to nTwistType. If IG_TWIST_RANDOM is chosen then each block is rotated one of the directions selected randomly.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

See IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.11.13 IG_FX_watermark

This function is used to produce a watermark like effect.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_FX_watermark (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const HIGEAR hWatermark
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to watermark.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to be processed. Set = NULL for whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
hWatermark	const HIGEAR	8-bit sign centered image to use for watermarking.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette;
Images that have a Grayscale LUT attached to them.

Example:

```
HIGEAR hIGear;     /* Handle of image to apply watermark to   */
HIGEAR hWMarkImage; /* Handle of watermark image   */
IG_FX_watermark ( hIGear, NULL, hWMarkImage );
```

Remarks:

8-bit sign centered image hWatermark is scaled to match image hIGear and is added to it.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.12 General Image Functions

This section provides information about the General Image group of functions.

- IG_image_batch_convert
- IG_image_bits_per_channel_get
- IG_image_compression_type_get
- IG_image_control_get
- IG_image_control_set
- IG_image_convert
- IG_image_create
- IG_image_create_alpha
- IG_image_create_DIB
- IG_image_create_DIB_ex
- IG_image_create_empty
- IG_image_delete
- IG_image_dimensions_get
- IG_image_duplicate
- IG_image_grayscale_LUT_copy_get
- IG_image_grayscale_LUT_exists
- IG_image_grayscale_LUT_update_from
- IG_image_is_gray
- IG_image_is_PDF
- IG_image_is_signed_get
- IG_image_is_signed_set
- IG_image_is_valid
- IG_image_orientation_get
- IG_image_orientation_set
- IG_image_resolution_get
- IG_image_resolution_set
- IG_image_savelist_get

1.3.1.2.12.1 IG_image_batch_convert

This function is designed to convert a specified set of files from one ImageGear-supported image format type to another.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_batch_convert(
        LPAT_SRCINFO lpSrcInfo,
        LPAT_DSTINFO lpDstInfo,
        const LPSTR lpcszLogFileName
);
```

Arguments:

Name	Type	Description
lpSrcInfo	LPAT_SRCINFO	A long pointer to a structure of type AT_SRCINFO through which you supply ImageGear with the source directory and format type of the files to be converted. See details below.
lpDstInfo	LPAT_DSTINFO	A long pointer to a structure of type AT_DSTINFO in which you supply ImageGear with the destination directory, format type, and naming convention for the newly converted files. See details below.
lpcszLogFileName	const LPSTR	Set this string to a filename for a log file to be generated. The log file will contain a list of files successfully converted, and any images that caused errors. If you do not need a log file, set this to NULL.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT   nErrcount;
LPAT_DSTINFO  DstInfo;
LPAT_SRCINFO  SrcInfo;
SrcInfo.lpcszSrcDir = "c:\public\source\images";
SrcInfo.lpcszSrcFilter = "*.bmp;*.tif";
DstInfo.lpcszDstDir = "c:\public\richard\joe\rich";
DstInfo.DstNamingConv = IG_BATCH_USE_SRC_NAME;
DstInfo.DstSaveType = IG_SAVE_TIF_UNCOMP;
/* convert all .bmp and .tif files from c:\public\source\images to TIFF uncompressed
images and store the coverted images to the destination directory
c:\public\richard\joe\rich, using the TIFF uncompressed file format*/
nErrcount = IG_image_batch_convert(&SrcInfo, &DstInfo," c:\\public\\log.txt");
```

Remarks:

The function takes three parameters: the source information (a structure of type AT_SRCINFO), the destination information (a structure of type AT_DSTINFO), and a const LPSTR to which you specify the path\filename of the log file to create.

AT_SRCINFO is a structure that contains the source file information:

```
typedef struct tag AT_SRCINFO
```

```
{
   LPSTR lpcszSrcDir; /* source dir from which files will be gathered */
   LPSTR lpcszSrcFilter;/* source filter for files contained on the
                                                            source dir*/
}AT_SRCINFO, FAR *LPAT_SRCINFO;
```

The source information structure must be completed entirely; no fields may be left out. The lpcszSrcDir should be a NULL-terminated string of characters representing the source directory, or where the images that are to be converted will be read from (ex. "c:\public\source\images"). The lpcszSrcFilter should be a NULL-terminated string of characters that represents what type of images should be converted. You may specify more than one type of image to be converted, e.g."*.bmp;*.tif". Each individual filter should be separated by a semicolon.

The LPAT_DSTINFO is a structure that contains the file destination information.

```
typedef struct tag AT_DSTINFO
{
LPSTR lpcszDstDir;      /*destination directory */
AT_LMODE DstOptions;       /*destination naming convention*/
AT_LMODE DstSaveType;      /*destination save type */
} AT_DSTINFO, FAR *LPAT_DSTINFO;
```

The destination information structure must be completed entirely; no fields may be left out. The lpcszDstDir argument should be a NULL-terminated string of characters that represents where the images that are to be converted will be stored after they are converted. This directory may or may not exist at the time this function is called. If the directory does not exist this function will create it (ex. "c:\public\destination\images"). If the source image is a multi-page image and the destination format type supports multiple pages, a new multi-page file will be created. If the source image is a multi-page image and the destination save type does not support multiple pages, the resulting destination image file or files will be determined by the naming convention that you supply to the AT_DSTINFO structure. If the naming convention IG_BATCH_USE_SRC_NAME is used, there will be one destination file which is continually overwritten by each subsequent page, and will ultimately contain only the last page of the original source file.

The DstNamingConv should contain one of the predefined constants from accucnst.h, in the section under "*Batch Naming Conventions*". Currently, the following naming conventions are available:

IG_BATCH_USE_SRC_NAME: This naming convention will use the source file name, remove the extension and replace it with the new save type default extension for naming each of the converted images. The DstSaveType should be one of the save types defined in the accucnst.h file (ex.IG_SAVE_TIF_UNCOMP).

You must enter a valid path and filename when you set lpczLogFileName.where the path that you specify already exists. If lpcszLogFileName is set to a valid filename, any pre-existing file will be overwritten with the new conversion information. If there is no such file, it will be created. The format of the log file, lpcszLogFileName, if the user has chosen to generate one, will be as follows for image files that are successfully converted:

```
Image:<Src file name> <src format, src comp>  Converted: <Dst file name> <dst format, dst
comp>
Image:<Src file name> <src format, src comp>  Converted: <Dst file name> <dst format, dst
comp>
Image:<Src file name> <src format, src comp>  Converted: <Dst file name> <dst format, dst
comp>
Image:<Src file name> <src format, src comp>  Converted: <Dst file name> <dst format, dst
comp>
...
```

If any image files cause errors during the convert, the format of the log file will be as follows:

```
Image:<Src file name> <src format, src comp> <error type> <error code number> >
Image:<Src file name> <src format, src comp> <error type> <error code number> >
Image:<Src file name> <src format, src comp> <error type> <error code number> >
```

1.3.1.2.12.2 IG_image_bits_per_channel_get

This function gets the number of bits allocated for each pixel channel in an image: 8, 16, or 32.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_bits_per_channel_get(
        HIGEAR hIGear,
        AT_INT* lpBitsPerChannel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpBitsPerChannel	AT_INT*	Returned number of allocated bits for each pixel channel.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
AT_INT nBits;           /* Number of bits per channel */
nErrcount = IG_image_bits_per_channel_get(hImage, &nBits);
```

Remarks:

This is not the same as the bit depth of a channel. The number of bits used for a pixel channel may be less than the number of bits allocated.

1.3.1.2.12.3 IG_image_compression_type_get

This function returns the compression type used for storing the image indicated by hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_compression_type_get(
       HIGEAR hIGear,
       LPDWORD lpCompression
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpCompression	LPDWORD	Pointer to a variable which will receive the compression type. See enumIGBiCompression for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

ImageGear currently uses only two types of image storage (compression): "Run Ends" and "Standard". ImageGear always uses "Run Ends" compression for 1-bit images, and "Standard" (uncompressed) format for all other types of images. If the function returns any value other than IG_BI_RLE and IG_BI_EMPTY, this means that the DIB uses Standard storage format. This behavior is preserved for compatibility with previous versions of ImageGear.

1.3.1.2.12.4 IG_image_control_get

This function has been deprecated and will be removed from the public API in a future release. Please use IG_fltr_ctrl_get instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_control_get(
       AT_MODE nOption,
       LPVOID lpData
);
```

Arguments:

Name	Type	Description
nOption	AT_MODE	Image option ID.
lpData	LPVOID	Reference to option data associated with the option ID.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function has been deprecated and will be removed from the public API in a future release. Please use IG_fltr_ctrl_get instead.

This function retrieves the properties associated with the specified option ID.

See enumControlOpt for further information on image option IDs.

To determine the number of errors currently on the error stack use IG_error_check. After fetching all error information you need using IG_error_get, use IG_error_clear to clear the stack.

1.3.1.2.12.5 IG_image_control_set

This function has been deprecated and will be removed from the public API in a future release. Please use IG_fltr_ctrl_set instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_control_set(
       AT_MODE nOption,
       LPVOID lpData
);
```

Arguments:

Name	Type	Description
nOption	AT_MODE	Image option ID.
lpData	LPVOID	Reference to option data to associate with the option ID.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function has been deprecated and will be removed from the public API in a future release. Please use IG_fltr_ctrl_set instead.

This function sets the properties associated with the specified option ID.

See enumControlOpt for further information on image option IDs.

To determine the number of errors currently on the error stack use IG_error_check. After fetching all error information you need using IG_error_get, use IG_error_clear to clear the stack.

1.3.1.2.12.6 IG_image_convert

This function allows you to transform image file without decoding it completely and avoiding the need to load it into a memory.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_convert(
        char* lpszSrcFileName,
        char* lpszDstFileName,
        AT_LMODE lFormatType,
        AT_LMODE lCommand,
        AT_LMODE lOptions
);
```

Arguments:

Name	Type	Description
lpszSrcFileName	char*	The filename of the source file.
lpszDstFileName	char*	The filename of the destination file.
lFormatType	AT_LMODE	The format type of output image to save.
lCommand	AT_LMODE	The type of operation to perform (IG_CONVERT_ type constant).
lOptions	AT_LMODE	Conversion options (bit mask).

Return Value:

Number of errors occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

The following types of transformations are currently supported:

- Lossless conversion of JPEG compressed image files
- Conversion between PDF and PostScript formats

PostScript format is not supported on MacOS X platform.

This function does not process image's metadata. This is responsibility of the user. See filter sample for an example of working with metadata.

Lossless Conversion of JPEG Compressed Image Files

This function allows you to apply certain operations on JPEG compressed image files, such as rotation, flipping, etc, without degradation of image quality. It can be used also for lossless conversion between JFIF JPEG and EXIF JPEG file formats, and for adding a thumbnail to a JFIF or EXIF file.

Transformation is done on the DCT coefficients rather than on decompressed pixels, so the lossy decompression/compression stages are not involved.

In contrast, the usual way (to load, rotate and save image) results in significant image degradation, especially when a high compression rate is used.

This function can be useful for converting photographic images between portrait and landscape layouts.

The following lossless operations (lCommand argument) are supported:

IG_CONVERT_NONE	No conversion.
IG_CONVERT_ROTATE_90	Rotate 90 degrees.

IG_CONVERT_ROTATE_180	Rotate 180 degrees.
IG_CONVERT_ROTATE_270	Rotate 270 degrees.
IG_CONVERT_FLIP_HORIZONTAL	Flip horizontal.
IG_CONVERT_FLIP_VERTICAL	Flip vertical.
IG_CONVERT_TRANSPOSE	Flip about upper left - lower right diagonal.
IG_CONVERT_TRANSVERSE	Flip about upper right - lower left diagonal.

IG_CONVERT_NONE mode can be used for converting between JFIF JPEG and EXIF JPEG format, or for adding a thumbnail to the image.

lOption parameter is a bit mask. Only one bit flag is supported:

IG_CONVERT_OPTION_TRIM = 1

An inherent limitation of such conversions is that the source image dimensions must be multiples of the DCT matrix size (typically 8 or 16) to preserve the entire image. If they are not, the remaining pixels at the right and/or bottom are undefined after transform. The function fills them with a mirror projection of the preceding pixels. This may work well enough for many photographic pictures. If you prefer not to keep the mirrored edge, set lOption parameter to IG_CONVERT_OPTION_TRIM. With this option set, the function will trim result image dimensions to a multiple of DCT size. In particular:

IG_CONVERT_NONE	Resulting image dimensions will not be modified.
IG_CONVERT_ROTATE_90	Resulting image width can be trimmed.
IG_CONVERT_ROTATE_180	Resulting image width and height can be trimmed.
IG_CONVERT_ROTATE_270	Resulting image height can be trimmed.
IG_CONVERT_FLIP_HORIZONTAL	Resulting image width can be trimmed.
IG_CONVERT_FLIP_VERTICAL	Resulting image height can be trimmed.
IG_CONVERT_TRANSPOSE	Resulting image dimensions will not be modified.
IG_CONVERT_TRANSVERSE	Resulting image width and height can be trimmed.

The following formats are supported as both source and destination: JFIF-JPEG, EXIF-JPEG. Only Lossy and Progressive compressions are supported. If any other format is used for either source or destination, the function will return an error.

If lFormatType == IG_FORMAT_UNKNOWN, the function will recognize source file format and use it for the destination file to save.

This function also converts the image's thumbnail, if it is present. If the thumbnail is JPEG compressed, it will be converted without degradation of quality. If the source image does not contain a thumbnail, and destination filter's "SAVE_THUMBNAIL" control parameter is set to TRUE, the function will create a thumbnail from the source image.

Conversion between PDF and PostScript Formats

This function can also be used for conversion of the entire document from PDF to PostScript or back. The function's arguments should be set as follows:

lpszSrcFileName	Name of the input PDF or PS document to convert.
lpszDstFileName	Name of the output PDF or PS document.
lFormatType	Save format, either IG_FORMAT_PDF or IG_FORMAT_POSTSCRIPT.
lCommand	IG_CONVERT_NONE.
lOptions	Not used, set to 0.

1.3.1.2.12.7 IG_image_create

This function creates a new image according to DIB information stored in a DIB info object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create(
       HIGDIBINFO hDIB,
       HIGEAR* lphIGear
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info object with parameters used to create image.
lphIGear	HIGEAR*	Returned HIGEAR handle of created image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
    // Create a new image with the same parameters as an existing image
    AT_ERRCOUNT nErrcount;  // Number of errors on stack
    HIGEAR hIGear;          // Handle of image
    HIGDIBINFO hDIB;        // DIB info handle of image
    HIGEAR hIGearCreated;   // Handle of created image

    // Load image file "picture.bmp" from working directory
    nErrcount = IG_load_file("picture.bmp", &hIGear);
    if(nErrcount == 0)
    {
        // Get DIB info of the existion image
        // DIB info can be also created and filled in manually
        nErrcount = IG_image_DIB_info_get(hIGear, &hDIB);
        if(nErrcount == 0)
        {
            nErrcount = IG_image_create(hDIB, &hIGearCreated);
            // Destroy DIB info
            IG_DIB_info_delete(hDIB);
            // ...
            // Destroy the image
            IG_image_delete(hIGearCreated);
        }
        // Destroy the source image
        IG_image_delete(hIGear);
    }
```

Remarks:

Pixel data is allocated and initialized to black.

1.3.1.2.12.8 IG_image_create_alpha

This function has been deprecated and will be removed from the public API in a future release. Please use IG_image_create, IG_image_channel_add, and IG_image_colorspace_convert instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create_alpha (
        HIGEAR hIGear,
        HIGEAR hIBackgrnd,
        AT_MODE nCreateMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
hIBackgrnd	HIGEAR	HIGEAR handle to a background image.
nCreateMode	AT_MODE	An integer value of type AT_MODE that tells ImageGear what bit depth the alpha channel should have. The possible settings for this variable, which are defined in accucnst.h are: IG_ALPHA_CREATE_1 and IG_ALPHA_CREATE_8.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear,  /* HIGEAR handles of images */
HIGEAR hIBackgrnd;
AT_ERRCOUNT    nErrcount;  /* Tally of ImageGear errors on the stack*/
nErrcount = IG_load_file ("Picture1.tga", &hIGear);
nErrcount = IG_load_file( "Picture2".bmp", &hIBackgrnd);
nErrcount = IG_image_create_alpha( hIGear, hIBackgrnd, IG_ALPHA_CREATE_8);
```

Remarks:

This function creates an alpha channel in the image hIGear, based on the data found in hIBackgrnd.

The height and width of hIBackgrnd must not be less than the height and width of hIGear. If there is already an alpha channel in the image, it will be replaced. Here is the formula by which the alpha channel data is calculated (where I2 stands for second image):

A = (I2 - Back) / (abs(I2 - back) - back);

If hIBackgrnd is a 1-bit image, you should set nCreateMode to IG_ALPHA_CREATE_1. When hIGear is displayed, this data will act as overlay data, where the 2 possible values for each bit of overlay data will determine whether the pixel is displayed or is made transparent, so that whatever is in the background will show through.

If hIBackgrnd is an 8-bit image, you should set nCreateMode to IG_ALPHA_CREATE_8. This will add 8 bits (with 256 possible values) of alpha data for each pixel of hIGear.

The image must support the storage of alpha data. Targa (*.tga) is an example of one that does. In a 24-bit Targa image, each pixel is stored to 32 bits, where the extra 8 bits may be used for alpha data.

1.3.1.2.12.9 IG_image_create_DIB

Please use the new upgraded function IG_image_create_DIB_ex().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create_DIB(
        AT_DIMENSION nWidth,
        AT_DIMENSION nHeight,
        UINT nBitsPerPixel,
        LPAT_DIB lpDIB,
        LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
nWidth	AT_DIMENSION	Set to the width that the image will be, in pixels. If the DIB already exists (lpDIB <> NULL), this value will be ignored.
nHeight	AT_DIMENSION	Set to the height that the image will be (number of rows). If the DIB already exists (lpDIB <> NULL), this value will be ignored.
nBitsPerPixel	UINT	Set to the bit depth of the new DIB. If the DIB already exists (lpDIB <> NULL), this value will be ignored .
lpDIB	LPAT_DIB	Far pointer to a DIB to copy, or NULL if creating an empty DIB. See the tip below. If this parameter is not NULL, it must be a valid pointer to the uncompressed bitmap. For example, the biCompression field of lpDIB can be either: IG_BI_RGB = 0 or IG_BI_GRAYSCALE = 503.
LphIGear	LPHIGEAR	A far pointer that returns a HIGEAR handle for the DIB just created.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB – 1, 4, 8 bpp;
Grayscale – 9…16 bpp;
RGB – 24 bpp;
CMYK – 32 bpp.

This function is only kept for backward compatibility reasons. Please use IG_image_DIB_import or IG_image_create instead.

Example:

```
(See also the example for function IG_dspl_DDB_import).
HIGEAR  hIGearNew;       /* Will be handle of new empty DIB */
AT_DIMENSION  nWid, nHi;       /* Dimensions for empty DIB  */
UINT    Bpp;              /* Bits per pixel for empty DIB */
AT_ERRCOUNT   nErrCount;       /* Count of errors put on stack */
HIGEAR  hIGearCopy;      /* Will be handle of new copied DIB */
char FAR  *lpExistingDIB; /* Holds address of an existing DIB */
/* Create an empty 500 x 300 x 16 bits per pixel DIB:   */
nWid  =  500;  nHi = 300;   /* Create a 500 pixel x 300 row DIB
*/
nBpp  =  16;                /* 16 Supported Raster Image Formats:  */
nErrCount =  IG_image_create_DIB (nWid, nHi, nBpp, NULL,
&hIGearNew);
if ( nErrs ) { ...}         /* Process any errors */
 ...
/* Copy DIB at *lpExistingDIB, creating HIGEAR image hIGearCopy:
*/
nErrCount =  IG_image_create_DIB (0, 0, 0,  (LPAT_DIB)
lpExistingDIB, &hIGearCopy);
if ( nErrs ) { ...}         /* Process any errors */
```

Remarks:

The functionality of this API call has been upgraded and supported by the new function IG_image_create_DIB_ex(). The reason that this new function has been created is that the old function cannot support 16-bit DIBs. In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

This function creates a new DIB and returns you its HIGEAR handle. If the FAR pointer lpDIB = NULL, an empty DIB is created using arguments nWidth, nHeight, and nBitsPerPixel. If lpDIB is not NULL, it should be a FAR pointer to an existing DIB which is to be copied. The DIB to be copied need not have a HIGEAR handle associated with it. The width, height, and Bits Per Pixel will be copied from the existing DIB; arguments nWidth, nHeight, and nBitsPerPixel will be ignored.

If you have an existing DIB which you simply want to give a HIGEAR handle to, use function IG_image_DIB_import(), which does not make a copy of the DIB.

If the lpDIB parameter is not NULL, then it must be a valid pointer to the uncompressed bitmap, that is the biCompression field of the lpDIB structure can be either IG_BI_RGB = 0 or IG_BI_GRAYSCALE = 503.

If you set lpDIB to NULL in order to create an empty DIB, the DIB palette will not be initialized. You will have to initialize it yourself. If you do not, the image will be displayed as all black.
Each raster in the DIB data must be padded to 32 bits. ImageGear does not support a top-down DIB (where biHeight is negative).

1.3.1.2.12.10 IG_image_create_DIB_ex

This function creates a new DIB and returns you its HIGEAR handle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create_DIB_ex(
        AT_DIMENSION nWidth,
        AT_DIMENSION nHeight,
        UINT nBitsPerPixel,
        AT_LMODE lCompression,
        LPAT_DIB lpDIB,
        LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
nWidth	AT_DIMENSION	Set to the width that the image will be, in pixels. If the DIB already exists (lpDIB <> NULL), this value will be ignored.
nHeight	AT_DIMENSION	Set to the height that the image will be (number of rows). If the DIB already exists (lpDIB <> NULL), this value will be ignored.
nBitsPerPixel	UINT	Set to the bit depth of the new DIB. If the DIB already exists (lpDIB <> NULL), this value will be ignored.
lCompression	AT_LMODE	Set to the type of pixel storage format you would like used in the new DIB. Currently, there are three options: ● IG_BI_RGB - for standard Windows DIB pixel storage. ● IG_BI_GRAYSCALE - for 16-bitgrayscale DIB pixel storage. ● IG_BI_CMYK - for 32-bit CMYK DIB pixel storage. **This variable is named lCompression because it is used to set up the biCompression field of the DIB header.
lpDIB	LPAT_DIB	Far pointer to a DIB to copy, or NULL if creating an empty DIB. See the tip below. If this parameter is not NULL, it must be a valid pointer to the uncompressed bitmap. For example, the biCompression field of lpDIB can be either: IG_BI_RGB = 0 or IG_BI_GRAYSCALE = 503.
LphIGear	LPHIGEAR	A far pointer that returns a HIGEAR handle for the DIB just created.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB – 1, 4, 8 bpp;
Grayscale – 9…16 bpp;
RGB – 24 bpp;
CMYK – 32 bpp.

This function is only kept for backward compatibility reasons. Please use IG_image_DIB_import or IG_image_create instead.

Example:

```
HIGEAR hIGearNew = NULL; // Will be handle of new empty DIB
AT_DIMENSION nWidth=0, nHight=0; // Dimensions for empty DIB
```

```
UINT nBpp = 0; // Bits per pixel for empty DIB
AT_LMODE nCompression = IG_COMPRESSION_NONE;
AT_ERRCOUNT nErrCount = 0; // Count of errors put on stack
HIGEAR hIGearCopy = NULL; // Will be handle of new copied DIB
char FAR *lpExistingDIB = NULL; // Holds address of an existing DIB
// Create an empty 500 x 300 x 16 bits per pixel DIB
nWidth = 500; nHight = 300; nBpp = 16;
nErrCount = IG_image_create_DIB_ex (nWidth, nHight, nBpp, nCompression, NULL, &hIGearNew);
if( nErrCount ) //Process any errors
// Copy DIB at *lpExistingDIB, creating HIGEAR image hIGearCopy
nErrCount = IG_image_create_DIB_ex (0, 0, 0, 0, (LPAT_DIB) lpExistingDIB, &hIGearCopy);
if( nErrCount ) //Process any errors
```

Remarks:

If the FAR pointer lpDIB = NULL, an empty DIB is created using arguments nWidth, nHeight, and nBitsPerPixel. If lpDIB is not NULL, it should be a FAR pointer to an existing DIB which is to be copied. The DIB to be copied need not have a HIGEAR handle associated with it. The width, height, and Bits Per Pixel will be copied from the existing DIB; arguments nWidth, nHeight, and nBitsPerPixel will be ignored.

If the lpDIB parameter is not NULL, then it must be a valid pointer to the uncompressed bitmap, that is the biCompression field of the lpDIB structure can be either IG_BI_RGB= 0 or IG_BI_GRAYSCALE= 503.

If you set lpDIB to NULL in order to create an empty DIB, the DIB palette will not be initialized. You will have to initialize it yourself. If you do not, the image will be displayed as all black.

Each raster in the DIB data must be padded to 32 bits. ImageGear does not support a top-down DIB (where biHeight is negative).

1.3.1.2.12.11 IG_image_create_empty

This function creates a new empty image that does not have pixel data allocated.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_create_empty(
       HIGDIBINFO hDIB,
       HIGEAR* lphIGear
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	DIB info object with parameters used to create image.
lphIGear	HIGEAR*	Returned HIGEAR handle of created image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
    // Create a new image with the same parameters as an existing image,
    // but with no pixel data allocated
    AT_ERRCOUNT nErrcount;  // Number of errors on stack
    HIGEAR hIGear;          // Handle of image
    HIGDIBINFO hDIB;        // DIB info handle of image
    HIGEAR hIGearCreated;   // Handle of created image

    // Load image file "picture.bmp" from working directory
    nErrcount = IG_load_file("picture.bmp", &hIGear);
    if(nErrcount == 0)
    {
        // Get DIB info of the existion image
        // DIB info can be also created and filled in manually
        nErrcount = IG_image_DIB_info_get(hIGear, &hDIB);
        if(nErrcount == 0)
        {
            nErrcount = IG_image_create_empty(hDIB, &hIGearCreated);
            // Destroy DIB info
            IG_DIB_info_delete(hDIB);
            // ...
            // Destroy the image
            IG_image_delete(hIGearCreated);
        }
        // Destroy the source image
        IG_image_delete(hIGear);
    }
```

1.3.1.2.12.12 IG_image_delete

This function deletes the HIGEAR handle and all memory associated with it.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_delete(
   HIGEAR hIGear
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to delete.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
    // Create a new image with the same parameters as an existing image,
    // but with no pixel data allocated
    AT_ERRCOUNT nErrcount;  // Number of errors on stack
    HIGEAR hIGear;          // Handle of image

    // Load image file "picture.bmp" from working directory
    nErrcount = IG_load_file("picture.bmp", &hIGear);
    if(nErrcount == 0)
    {
        // ...
        // Destroy the image
        IG_image_delete(hIGear);
    }
```

This function also frees the memory associated with the image's DIB, if ImageGear allocated the DIB memory. If ImageGear did not allocate the DIB memory, the DIB continues to exist, and it is the responsibility of your application to free this memory when done with it.

1.3.1.2.12.13 IG_image_dimensions_get

This function returns the width, height, and number of Bits Per Pixel, from the DIB of the image indicated by handle hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_dimensions_get(
       HIGEAR hIGear,
       LPAT_DIMENSION lpWidth,
       LPAT_DIMENSION lpHeight,
       LPUINT lpBitsPerPixel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle of image.
lpWidth	LPAT_DIMENSION	Pointer to a variable which will receive image width (number of pixels per row).
lpHeight	LPAT_DIMENSION	Pointer to a variable which will receive image height (number of rows).
lpBitsPerPixel	LPUINT	Pointer to a variable which will receive image bit depth (number of Bits Per Pixel).

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
AT_DIMENSION nWidth, nHeight;  // Will hold returned width and height
UINT nBpp;                // Will hold returned bits per pixel

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_image_dimensions_get(hIGear, &nWidth, &nHeight, &nBpp);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

"Bits per Pixel" parameter does not uniquely identify the image pixel format. For example, a 8-bits per channel CMYK image and a 8-bits per channel RGBA image will have the same "Bits per Pixel" value of 32. Please use IG_image_channel_count_get, IG_image_channel_depth_get, IG_image_channel_depths_get, IG_image_colorspace_get or IG_image_DIB_info_get to obtain more specific information.

1.3.1.2.12.14 IG_image_duplicate

This function creates an exact duplicate of the current HIGEAR image, and returns the handle to the new image to you in lphIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_duplicate (
        HIGEAR hIGear,
        LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the current image to be duplicated.
lphIGear	LPHIGEAR	A far pointer in which the handle of the new, duplicate image is returned.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;/* HIGEAR handle of image  */
HIGEAR lphIGear;  /* HIGEAR handle to new duplicate image   */
IG_image_duplicate(hIGear, &lphIGear);
```

1.3.1.2.12.15 IG_image_grayscale_LUT_copy_get

This function returns a copy of image grayscale LUT, if it exists.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_grayscale_LUT_copy_get(
        HIGEAR hIGear,
        HIGLUT* lpLUT
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear handle.
lpLUT	HIGLUT*	New LUT handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Currently, grayscale LUT is only taken into account for 2…16 bpp Grayscale images.

1.3.1.2.12.16 IG_image_grayscale_LUT_exists

This function checks whether HIGEAR has a grayscale LUT attached.

Declaration:

```
AT_BOOL ACCUAPI IG_image_grayscale_LUT_exists(
        HIGEAR hIGear
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Currently, grayscale LUT is only taken into account for 2…16 bpp Grayscale images.

1.3.1.2.12.17 IG_image_grayscale_LUT_update_from

This function updates (creates if not present) a grayscale LUT for the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_grayscale_LUT_update_from(
        HIGEAR hIGear,
        HIGLUT lut
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle.
lut	HIGLUT	LUT handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Currently, grayscale LUT is only taken into account for 2…16 bpp Grayscale images.

1.3.1.2.12.18 IG_image_is_gray

This function is called to determine if an image is a grayscale image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_is_gray(
       HIGEAR hIGear,
       LPAT_BOOL lpIsImageGray
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpIsImageGray	LPAT_BOOL	Pointer to a variable which will be overwritten with TRUE if the image is grayscale, and with FALSE if it is not grayscale.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
AT_BOOL bItsGray;        // Will be set = TRUE if grayscale

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_image_is_gray(hIGear, &bItsGray);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The function considers the image to be grayscale, if either of the following is true:

- Image has IG_COLOR_SPACE_ID_Gy (grayscale) colorspace.
- Image has IG_COLOR_SPACE_ID_I (indexed) colorspace, has more than 1 bit per pixel, all of its palette entries are grayscale (R[i] = G[i] = B[i]), and the palette is either non-decreasing (R[i] >= R[i-1] for all i>0) or non-increasing (R[i] <= R[i-1] for all i>0).
- Image has IG_COLOR_SPACE_ID_RGB colorspace, and all image pixels are grayscale: R = G = B.
- Color channels of the image satisfy one of the requirements listed above, and the image also has Alpha, Premultiplied Alpha or Extra channels.

FALSE is returned for all 1-bit indexed images, even if the palette contains two shades of gray.

1.3.1.2.12.19 IG_image_is_PDF

This function returns TRUE if the image is PDF.

Declaration:

```
AT_BOOL ACCUAPI IG_image_is_PDF( HIGEAR hIGear);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.

Return Value:

Returns TRUE if the image is PDF; FALSE - otherwise.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

1.3.1.2.12.20 IG_image_is_signed_get

This function returns a boolean value indicating whether the image pixel data is signed or unsigned.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_is_signed_get(
       HIGEAR hIGear,
       LPAT_BOOL lpbSigned
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpbSigned	AT_BOOL	Indicates whether or not a grayscale image should be treated as signed.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
AT_BOOL bItsSigned;        // Will be set = TRUE if signed

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_image_is_signed_get(hIGear, &bItsSigned);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

Although ImageGear allows getting and setting the Signed flag from/to images of any colorspaces, except for 1-bit per pixel images, it only takes this flag into account for images that have IG_COLOR_SPACE_ID_Gy colorspace.

Several image file formats, such as DICOM and JPEG2K, allow specifying image pixels as signed or unsigned. If the file format does not specify whether the pixels are signed or unsigned, ImageGear assumes they are unsigned.

If HIGEAR image is signed, and an attempt is made to save it to a file format that does not support Signed images, the Signed flag is ignored.

1.3.1.2.12.21 IG_image_is_signed_set

This function sets a boolean value that specifies whether the image pixel data is signed or unsigned.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_is_signed_set(
       HIGEAR hIGear,
       AT_BOOL bSigned
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
bSigned	AT_BOOL	Indicates whether the image should be treated as signed or unsigned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB - 1 bpp;
- Grayscale - 1 bpp.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_image_is_signed_set(hIGear, TRUE);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The Signed flag affects the image display. If an image is unsigned, and does not have any display LUTs attached, pixel intensity value of 0 is the minimal intensity, so it is displayed as black. If the image is signed, 0 is the middle intensity, so it is displayed as 50% gray.

Although ImageGear allows getting and setting the Signed flag from/to images of any colorspaces, except for 1-bit per pixel images, it only takes this flag into account for images that have IG_COLOR_SPACE_ID_Gy colorspace.

Several image file formats, such as DICOM and JPEG2K, allow specifying image pixels as signed or unsigned. If the file format does not specify whether the pixels are signed or unsigned, ImageGear assumes they are unsigned.

If HIGEAR image is signed, and an attempt is made to save it to a file format that does not support Signed images, the Signed flag is ignored.

This function does not modify the image pixel values. It only changes a flag attached to the image. Also, this function does not cause the image to be redrawn. Refer to IG_dspl_image_draw for how to display an image.

1.3.1.2.12.22 IG_image_is_valid

This function is called to determine if the HIGEAR variable hIGear contains a handle of a valid ImageGear image.

Declaration:

```
BOOL ACCUAPI IG_image_is_valid (HIGEAR hIGear);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.

Return Value:

This function returns TRUE if a hIGear contains a valid handle; FALSE otherwise.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;      /* Contains HIGEAR handle of image   */
if ( IG_image_is_valid ( hIGear ) )
        { IG_save_file ( hIGear, "picture.bmp", IG_SAVE_BMP_UNCOMP ); }
```

Remarks:

Note that the return-type of the function is BOOL, not AT_ERRCOUNT. TRUE is returned if the handle is valid and may be used as the HIGEAR argument in calls to other ImageGear functions.

1.3.1.2.12.23 IG_image_orientation_get

This function tells you the orientation of the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_orientation_get(
       HIGEAR hIGear,
       LPAT_MODE lpOrientation
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpOrientation	LPAT_MODE	A constant of type AT_MODE that will return the current orientation of the HIGEAR image. See enumOrientation for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
AT_MODE nOrientation;    // Image orientation

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_image_orientation_get(hIGear, &nOrientation);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The two most frequently used orientations are "Portrait" and "Landscape." However, in order to provide complete support for the TIFF file format, which defines eight image orientations, ImageGear interprets eight orientations. For an image with a "Portrait" orientation, this value would normally be IG_ORIENT_TOP_LEFT. For an image with "Landscape" orientation, this value would be either IG_ORIENT_RIGHT_TOP or IG_ORIENT_LEFT_BOTTOM. See enumOrientation for descriptions of all orientation modes.

Notice that the IG_ORIENT constants contain indicators of two directions. For IG_ORIENT_TOP_LEFT, the first direction is "TOP." This specifies the placement of row 0 of the image. The second direction is "LEFT", and this specifies the placement of column 0. Thus, IG_ORIENT_TOP_LEFT specifies that row 0 of the image stored in the file should be displayed at the top and column 0 should be displayed at the left. This is the normal orientation of most images. On the other hand, IG_ORIENT_LEFT_BOTTOM specifies that row 0 should be displayed at the left and column 0 at the bottom. For this to be true the image would have to be rotated 90 degrees counterclockwise.

The orientation setting is stored in the header structure of those file formats that support the storage of orientation information. The orientation setting tells how the image was intended to be displayed. In the example above, the bitmap image is not necessarily stored "sideways." When you find that it is intended to be displayed sideways, you could call IG_dspl_orientation_get to display it in its intended orientation, or call

IG_IP_rotate_multiple_90 to rearrange the actual bitmap data so that the image is actually stored sideways.

1.3.1.2.12.24 IG_image_orientation_set

This function sets the orientation of the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_orientation_set(
       HIGEAR hIGear,
       AT_MODE nOrientation
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nOrientation	AT_MODE	A constant of type AT_MODE that sets the orientation of the HIGEAR image. See enumOrientation for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_image_orientation_set(hIGear, IG_ORIENT_LEFT_BOTTOM);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The two most frequently used orientations are "Portrait" and "Landscape." However, in order to provide complete support for the TIFF file format, which defines eight image orientations, ImageGear interprets eight orientations. For an image with a "Portrait" orientation, this value would normally be IG_ORIENT_TOP_LEFT. For an image with "Landscape" orientation, this value would be either IG_ORIENT_RIGHT_TOP or IG_ORIENT_LEFT_BOTTOM. See enumOrientation for descriptions of all orientation modes.

Notice that the IG_ORIENT constants contain indicators of two directions. For IG_ORIENT_TOP_LEFT, the first direction is "TOP." This specifies the placement of row 0 of the image. The second direction is "LEFT", and this specifies the placement of column 0. Thus, IG_ORIENT_TOP_LEFT specifies that row 0 of the image stored in the file should be displayed at the top and column 0 should be displayed at the left. This is the normal orientation of most images. On the other hand, IG_ORIENT_LEFT_BOTTOM specifies that row 0 should be displayed at the left and column 0 at the bottom. For this to be true the image would have to be rotated 90 degrees counterclockwise.

The orientation setting is stored in the header structure of those file formats that support the storage of orientation information. The orientation setting tells how the image was intended to be displayed. In the last example above, the bitmap image is not necessarily stored "sideways." When you find that it is intended to be displayed sideways, you could call IG_dspl_orientation_get to display it in its intended orientation, or call IG_IP_rotate_multiple_90 to rearrange the actual bitmap data so that the image is actually stored sideways.

1.3.1.2.12.25 IG_image_resolution_get

This function retrieves the current resolution settings of the HIGEAR image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_resolution_get(
        HIGEAR hIGear,
        LPLONG lpXResNumerator,
        LPLONG lpXResDenominator,
        LPLONG lpYResNumerator,
        LPLONG lpYResDenominator,
        LPAT_MODE lpnUnits
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
lpXResNumerator	LPLONG	Pointer to a LONG variable that receives the x resolution numerator.
lpXResDenominator	LPLONG	Pointer to a LONG variable that receives the x resolution denominator.
lpYResNumerator	LPLONG	Pointer to a LONG variable that receives the y resolution numerator.
lpYResDenominator	LPLONG	Pointer to a LONG variable that receives the y resolution denominator.
lpnUnits	LPAT_MODE	Pointer to an AT_MODE variable that receives the resolution units. See enumIGResolutionUnits for possible values.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;
AT_RESOLUTION res;
AT_ERRCOUNT  nErrCount;

// Load the image
nErrCount = IG_load_file("picture.tif", &hIGear);

if (nErrCount == 0)
{
        // Obtain the image's resolution
        nErrCount = IG_image_resolution_get(hIGear, &res.xResNumerator,
&res.xResDenominator,
                &res.yResNumerator, &res.yResDenominator, &res.nUnits);

        // Change the current resolution setting to INCHES
        if (nErrCount == 0)
                nErrCount = IG_util_resolution_units_convert(&res, IG_RESOLUTION_INCHES);

        // Set the modified resolution
        if (nErrCount == 0)
                nErrCount = IG_image_resolution_set(hIGear, res.xResNumerator,
res.xResDenominator,
                 res.yResNumerator, res.yResDenominator, res.nUnits);

        // ...

        // Delete the image
        IG_image_delete(hIGear);
}
```

Remarks:

This function returns the resolution values and units of the HIGEAR image.

ImageGear stores resolution as a pair or rational numbers and a unit specification. This is the method used by several image file formats, which allows storing precise resolution values, rather than their double or float approximations. To set the X resolution of the image to 300 DPI, the numerator can be 300 and the denominator 1 (900 and 3 would also work).

Use IG_image_resolution_set to set image resolution.

Use IG_util_resolution_units_convert to convert resolution to different units.

An alternative way to obtain the image resolution is to get its DIB information using IG_image_DIB_info_get, and then get the resolution using IG_DIB_resolution_get.

1.3.1.2.12.26 IG_image_resolution_set

This function sets the resolution of the image referenced by hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_resolution_set(
        HIGEAR hIGear,
        LONG xResNumerator,
        LONG xResDenominator,
        LONG yResNumerator,
        LONG yResDenominator,
        AT_MODE nUnits
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
XResNumerator	LONG	Sets the x resolution numerator.
XResDenominator	LONG	Sets the x resolution denominator.
YResNumerator	LONG	Sets the y resolution numerator.
YResDenominator	LONG	Sets the y resolution denominator.
nUnits	AT_MODE	Sets the resolution units for the image. See enumIGResolutionUnits for possible values.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

See the example under the IG_image_resolution_get() function.

Remarks:

ImageGear stores resolution as a pair or rational numbers and a unit specification. This is the method used by several image file formats, which allows storing precise resolution values, rather than their double or float approximations. To set the X resolution of the image to 300 DPI, the numerator can be 300 and the denominator 1 (900 and 3 would also work).

When an image is saved to a file, resolution will be converted when necessary to match the units supported by the file format.

ImageGear uses resolution information when printing the image. The ratio of image resolutions (X and Y) can also be used by ImageGear when displaying the image. Use IG_dspl_PPM_correct_set to specify whether ImageGear should use the ratio of image resolutions when displaying and printing the image.

Changing these values does not alter the number of pixels or colors in the actual image in any way. Use IG_IP_resize() and IG_IP_crop() to resize or crop an image.

Use IG_image_resolution_get to obtain the image resolution information.

Use IG_util_resolution_units_convert to convert resolution to different units.

1.3.1.2.12.27 IG_image_savelist_get

This function prepares the list of constants corresponding to format and compression combinations available for saving of the specified image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_savelist_get(
   HIGEAR hIGear,
   LPAT_MODE lpnFilterList,
   UINT nFListSize,
   LPAT_LMODE lpSaveList,
   UINT nSListSize,
   LPUINT lpnSListCount
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Handle of the image to check the compression list against. If the value is not NULL, this function returns the list of enumIGSaveFormats values corresponding to saving formats (format and compression combinations) available for saving of the specified image. If the value is NULL, then the function returns the list of all currently supported saving formats for file formats specified by lpnFilterList. If both the hIGear and lpnFilterList are null, the function returns the list of all currently supported saving formats.
lpnFilterList	LPAT_MODE	Pointer to the list of format identifiers, which will be used in the save list. See enumIGFormats for possible values. If this parameter is NULL, then all currently supported formats will be used.
nFListSize	UINT	Array containing the number of elements if lpnFilterList is not NULL.
lpSaveList	LPAT_MODE	Array containing the returned saving format constants. You can set this value to NULL if you only need to obtain the total number of found saving formats.
nSListSize	UINT	Size of the lpSaveList array.
lpnSListCount	LPUINT	If the lpSaveList array is not NULL, this parameter returns the number of copied enumIGSaveFormats values. If lpSaveList is NULL, this parameter returns the total number of records.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrCount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
UINT nCount;            // Number of save formats
LPAT_LMODE lpSaveList;

// Load the image
nErrCount = IG_load_file("picture.tif", &hIGear);
if( nErrCount == 0 )
{
    // Get save formats count
    nErrCount = IG_image_savelist_get(hIGear, NULL, 0, NULL, 0, &nCount);
    // Allocate memory
    lpSaveList = (LPAT_LMODE)malloc( nCount*sizeof(AT_LMODE) );
    if( lpSaveList!=NULL )
    {
        // Get save list
        nErrCount = IG_image_savelist_get(hIGear, NULL, 0, lpSaveList, nCount, NULL);

        //...

        // Delete memory
        free(lpSaveList);
    }
    // Delete the image
    IG_image_delete(hIGear);
}
```

Remarks:

Records returned by the function are sorted alphabetically by their short names. Short names correspond to those returned by IG_fltr_info_get function.

This function works similarly to IG_fltr_compressionlist_get_ex, but it works with all formats supported by ImageGear rather than with a particular format. Values returned in the lpSaveList can be passed directly to ImageGear saving functions such as IG_fltr_save_file.

See also the section Using Format Filters API for Image Saving.

1.3.1.2.13 Global Control Parameter Functions

This section provides information about the Global Control Parameter group of functions.

- IG_gctrl_item_by_index_get
- IG_gctrl_item_count_get
- IG_gctrl_item_get
- IG_gctrl_item_id_get
- IG_gctrl_item_set

1.3.1.2.13.1 IG_gctrl_item_by_index_get

This function is like IG_gctrl_item_get(), but returns information about parameter determined by given index in array.

Declaration:

```
AT_BOOL  ACCUAPI  IG_gctrl_item_by_index_get(
        UINT nIndex,
        LPCHAR CtrlID,
        DWORD dwIDSize,
        LPAT_MODE lpnValType,
        LPVOID lpValue,
        DWORD dwValSize,
        LPDWORD lpdwValSize,
        LPCHAR lpTextInfo,
        DWORD dwTextBufSize,
        LPDWORD lpdwTextInfoSize
);
```

Arguments:

Name	**Type**	**Description**
nIndex	UINT	IN: Index of global control parameter in the array.
CtrlID	LPCHAR	OUT: The name of global control parameter.
dwIDSize	DWORD	IN: Size of CtrlID in bytes.
lpnValType	LPAT_MODE	OUT: The type of global control parameter value.
lpValue	LPVOID	OUT: Buffer where to copy control parameter value.
dwValSize	DWORD	IN: Size of lpValue buffer in bytes.
lpdwValSize	LPDWORD	OUT: Actual size of parameter value in bytes.
lpTextInfo	LPCHAR	OUT: Buffer where to copy text description of global control parameter.
dwTextBufSize	DWORD	IN: Size of lpTextInfo buffer in bytes.
lpdwTextInfoSize	LPDWORD	OUT: Actual size of the text description of parameter.

Return Value:

TRUE if global parameter with given name is found; FALSE if it does not exist.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

See also the section Working with Global Control Parameters.

1.3.1.2.13.2 IG_gctrl_item_count_get

This function returns total amount of global parameters in the list.

Declaration:

```
UINT  ACCUAPI  IG_gctrl_item_count_get();
```

Arguments:

None

Return Value:

Integer value - amount of global parameters in the list.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

See also the section Working with Global Control Parameters.

1.3.1.2.13.3 IG_gctrl_item_get

This function returns all information about global control parameter identified by name CtrlID.

Declaration:

```
AT_BOOL  ACCUAPI  IG_gctrl_item_get(
        LPCHAR CtrlID,
        LPAT_MODE lpnValType,
        LPVOID lpValue,
        DWORD dwValSize,
        LPDWORD lpdwValSize,
        LPCHAR lpTextInfo,
        DWORD dwTextBufSize,
        LPDWORD lpdwTextInfoSize
);
```

Arguments:

Name	Type	Description
CtrlID	LPCHAR	IN: The name of global control parameter.
lpnValType	LPAT_MODE	OUT: The type of global control parameter value.
lpValue	LPVOID	OUT: Buffer where to copy control parameter value.
dwValSize	DWORD	IN: Size of lpValue buffer in bytes.
lpdwValSize	LPDWORD	OUT: Actual size of parameter value in bytes.
lpTextInfo	LPCHAR	OUT: Buffer where to copy text description of global control parameter.
dwTextBufSize	DWORD	IN: Size of lpTextInfo buffer in bytes.
lpdwTextInfoSize	LPDWORD	OUT: Actual size of the text description of parameter.

Return Value:

TRUE if global parameter with given name is found; FALSE if it does not exist.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Returns TRUE if parameter with given name is found, and FALSE if parameter with given name does not exist.

See also the section Working with Global Control Parameters.

1.3.1.2.13.4 IG_gctrl_item_id_get

This function returns index of the global control parameter with given name in array.

Declaration:

```
AT_BOOL  ACCUAPI  IG_gctrl_item_id_get(
        UINT nIndex
        LPCHAR lpCtrlID,
        UINT nBufSize
);
```

Arguments:

Name	Type	Description
nIndex	UINT	OUT: An index of lpCtrlID global control parameter in the parameters array.
lpCtrlID	LPCHAR	IN: The name of global control parameter.
nBufSize	UINT	IN: Size of lpCtrlID buffer in bytes.

Return Value:

TRUE if global parameter with given name is found; FALSE if it does not exist.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

See also the section Working with Global Control Parameters.

1.3.1.2.13.5 IG_gctrl_item_set

This function sets the value to global control parameter.

Declaration:

```
AT_ERRCODE  ACCUAPI IG_gctrl_item_set(
        LPCHAR CtrlID,
        AT_MODE nValueType,
        LPVOID lpValue,
        DWORD dwValueSize,
        LPCHAR lpTextInfo
);
```

Arguments:

Name	Type	Description
CtrlID	LPCHAR	IN: The name of global control parameter in form "<GRPNAME>.<Param name>".
nValueType	AT_MODE	IN: The type of global control parameter value. Constant of kind AM_TID_...
lpValue	LPVOID	IN: Pointer to global control parameter value data.
dwValueSize	DWORD	IN: Size of global control parameter value data (size of buffer lpValue)
lpTextInfo	LPCHAR	IN: Text description of global control parameter value.

Return Value:

Return value is a code of last error or NULL if success.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If parameter with given name ControlID does not exist, then it is added. If control parameter with given value exists and callback function for it is not NULL then callback is called exactly after value is changed by this function. If lpTextInfo is not NULL, then previous value of this field is changed to new value, but if NULL then it is not changed.

See also the section Working with Global Control Parameters.

1.3.1.2.14 Image Blending Functions

This section provides information about the Image Blending group of functions.

- IG_image_blend_with_alpha

1.3.1.2.14.1 IG_image_blend_with_alpha

This function blends two images together using their alpha channels.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_blend_with_alpha(
        HIGEAR hSource,
        HIGEAR hAlpha,
        LPAT_RECT blendingArea
);
```

Arguments:

Name	Type	Description
hSource	HIGEAR	First image for blending. It may or may not contain an alpha channel.
hAlpha	HIGEAR	Second image for blending. It must contain an alpha channel.
blendingArea	LPAT_RECT	Rectange area of the first image for blending. NULL means the entire image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

hSource:

- Grayscale - 2..16 bpp;
- RGB - 6..48 bpp;
- GyA - 4..32 bpp;
- GyPA - 4..32 bpp;
- RGBA - 8..64 bpp;
- RGBPA - 8..64 bpc.

hAlpha:

- GyA - 4..32 bpp;
- GyPA - 4..32 bpp;
- RGBA - 8..64 bpp;
- RGBPA - 8..64 bpc.

Remarks:

hAlpha is blended over hSource, and the result is stored in hSource. hAlpha should contain an Alpha channel, otherwise, an error is returned. Color channels (all channels except Extra and Alpha/Premultiplied Alpha) of both images must have the same color space and the same bit depths.

If hSource does not contain an Alpha channel, hAlpha is composited over it using hSource as the background. The resulting image does not contain an Alpha channel. The following formulas are used:

- If Page2 is not pre-multiplied,
- I1 = I2 * A2 + I1 (1-A2)
- If Page2 is pre-multiplied,
- I1 = I2 + I1 (1-A2)

If hSource contains an Alpha channel, hSource and hAlpha are blended together. The resulting image contains an alpha channel. Blending two images, both of which contain an Alpha channel, can be interpreted as placing one semi-transparent film over another semi-transparent film. The result of applying such combined images to some background is the same as applying one image over background and then applying another image to the result. The code

- IG_image_blend_with_alpha(hImage1, hImage2, NULL);
- IG_image_blend_with_alpha(hBackground, hImage1, NULL);

will produce the same result as

- IG_image_blend_with_alpha(hBackground, hImage1, NULL);
- IG_image_blend_with_alpha(hBackground, hImage2, NULL);

The following pseudocode demonstrates the formulas used by the function:

- if (A1=A2=0)
- I1 = 0; A1 = 0;
- else
- If both Page1 and Page2 are not pre-multiplied,
- I1 = (I2*A2 + I1*A1*(1-A2)) / (A1 + A2 - A1*A2)
- A1 = A1 + A2 - A1*A2
- //Result is not pre-multiplied
- If Page1 is not pre-multiplied, and Page2 is pre-multiplied:
- I1 = (I2 + I1*A1*(1-A2)) / (A1 + A2 - A1*A2)
- A1 = A1 + A2 - A1*A2
- //Result is not pre-multiplied
- If Page1 is pre-multiplied, and Page2 is not:
- I1 = I2*A2 + I1*(1-A2)
- A1 = A1 + A2 - A1*A2
- //Result is pre-multiplied
- If both Page1 and Page2 are pre-multiplied:
- I1 = I2 + I1*(1-A2)
- A1 = A1 + A2 - A1*A2
- //Result is pre-multiplied

Alpha values are mapped to a float value between 0.0 and 1.0, where 0.0 means full transparency and 1.0 means full opaquity. Alpha value of 0 corresponds to float value of 0.0 and alpha value of 2^n-1 (where n is alpha channel's bit depth) corresponds to float value of 1.0.

This function does not process Extra Channels.

1.3.1.2.15 Image Channel Functions

This section provides information about the Image Channel group of functions.

- IG_image_channel_add
- IG_image_channel_copy_create
- IG_image_channel_count_get
- IG_image_channel_depth_get
- IG_image_channel_depths_get
- IG_image_channel_depths_change
- IG_image_channel_remove
- IG_image_channel_update
- IG_image_channels_combine
- IG_image_channels_separate

1.3.1.2.15.1 IG_image_channel_add

This function adds a new channel to an image at the specified position, populating the channel's image data from another channel.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_add(
        HIGEAR hIGear,
        AT_UINT position,
        LPCAT_CHANNEL_REF channel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to which to add channel.
position	AT_UINT	Index of where channel should be added.
channel	LPCAT_CHANNEL_REF	Location of channel to add.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
/* Add an "extra" channel to an image */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of destination image */
HIGEAR hExtra;          /* Handle of image to use as extra */
AT_CHANNEL_REF channel; /* Channel to add */
AT_INT nChannels;       /* Number of channels in dest. image */
channel.hImage = hExtra;
channel.uNumber = 0;
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_channel_add(hImage, nChannels, &channel);
nErrcount = IG_image_colorspace_get(hImage, &cs);
```

Remarks:

The position is an index into the number of channels starting at 0. channel specifies the location of the channel to add, which consists of the HIGEAR handle of the image containing the channel and the index of the channel within that image.

1.3.1.2.15.2 IG_image_channel_copy_create

This function creates a new image by copying a single channel from an existing image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_copy_create(
        LPCAT_CHANNEL_REF source,
        LPAT_CHANNEL_REF copy
);
```

Arguments:

Name	Type	Description
source	LPCAT_CHANNEL_REF	Location of source channel to copy.
copy	LPAT_CHANNEL_REF	Location of new image with copied channel.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
/* Extract the first alpha channel if one exists */
/* Otherwise, extract the first color channel */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
AT_CHANNEL_REF srcChan; /* Channel to copy */
AT_CHANNEL_REF dstChan; /* New image with copied channel */
enumIGColorSpaceIDs cs; /* Source image's color space */
AT_INT nColorChannels;  /* Number of color channels */
nErrcount = IG_load_file("alpha.tif", &hImage);
nErrcount = IG_image_colorspace_get(hImage, &cs);
nColorChannels = IG_util_colorspace_color_count_get(cs);
srcChan.hImage = hImage;
if (IG_util_colorspace_contains_alpha(cs))
    srcChan.uNumber = nColorChannels; /* First alpha channel */
else
    srcChan.uNumber = 0; /* First color channel */
nErrcount = IG_image_channel_copy_create(&srcChan, &dstChan);
hCopy = dstChan.hImage;
nErrcount = IG_save_file(dstChan.hImage, "alpha.bmp",
    IG_SAVE_BMP_UNCOMP);
nErrcount = IG_image_delete(hImage);
```

Remarks:

Specify in source the image and channel index to copy. If the copy is successful, copy will contain the HIGEAR handle of a newly allocated image containing the copied channel. The channel index in copy will always be set to 0. You are responsible for freeing the new image with IG_image_delete.

1.3.1.2.15.3 IG_image_channel_count_get

This function counts the image channels.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_count_get(
        HIGEAR hIGear,
        AT_INT* lpChannelCount
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpChannelCount	AT_INT*	Channel count of the image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

1.3.1.2.15.4 IG_image_channel_depth_get

This function returns the channel bit depth.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_depth_get(
        HIGEAR hIGear,
        AT_INT Index,
        AT_INT* lpChannelDepth
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
Index	AT_INT	Channel index on which to get info.
lpChannelDepth	AT_INT*	Bit depth of the channel.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

1.3.1.2.15.5 IG_image_channel_depths_get

This function returns an array of the channel bit depths.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_depths_get(
        HIGEAR hIGear,
        AT_INT ChannelCount,
        AT_INT* lpChannelDepths
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
ChannelCount	AT_INT	Length of the channel depths array to be filled.
lpChannelDepths	AT_INT*	Array of the channel depths to be filled.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

1.3.1.2.15.6 IG_image_channel_depths_change

This function changes the bit depths of the image channels to the specified depths.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_depths_change(
        HIGEAR hIGear,
        const AT_INT* newDepths,
        AT_MODE scaleMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
newDepths	const AT_INT*	New channel depths to set.
scaleMode	AT_MODE	Mode to use for scaling channel depths - must be one of the following: IG_DEPTH_CHANGE_SCALE or IG_DEPTH_CHANGE_NO_SCALE.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
/* Alter a 24-bit RGB image to have full 8-bit precision
   for green, but only 1-bit precision for red and blue */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* HIGEAR handle of image */
AT_INT depths[] = { 1, 8, 1 }; /* New channel depths */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_depths_change(hImage, depths,
    IG_DEPTH_CHANGE_SCALE);
nErrcount = IG_save_file(hImage, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
nErrcount = IG_image_delete(hImage);
```

Remarks:

If scaling is used, pixel data is scaled to match the new bit depths. Otherwise, pixel data will remain unchanged and will be interpreted as conforming to the new depths.

1.3.1.2.15.7 IG_image_channel_remove

This function removes the specified channel from the source image and shifts the remaining channels without transforming pixel data.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_remove(
        HIGEAR hIGear,
        AT_UINT position
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
position	AT_UINT	Index of channel to remove.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:

- Images with 1 channel.

Example:

```
/* Alter a 24-bit RGB image to use the red channel data
   for both red and blue channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
AT_CHANNEL_REF channel; /* Channel to add */
nErrcount = IG_load_file("test.jpg", &hImage);
channel.hImage = hImage;
channel.uNumber = 0;
nErrcount = IG_image_channel_add(hImage, 3, &channel);
nErrcount = IG_image_channel_remove(hImage, 2);
nErrcount = IG_save_file(hImage, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

1.3.1.2.15.8 IG_image_channel_update

This function copies pixel data from one channel to another channel.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channel_update(
        LPCAT_CHANNEL_REF channelToUpdateWith,
        LPCAT_CHANNEL_REF channelToBeUpdated
);
```

Arguments:

Name	Type	Description
channelToUpdateWith	LPCAT_CHANNEL_REF	Channel to use as source channel.
channelToBeUpdated	LPCAT_CHANNEL_REF	Channel to be replaced by source channel.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
/* Alter a 24-bit RGB image to use the red channel data
for both red and blue channels */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
AT_CHANNEL_REF chanSrc; /* Channel to update from */
AT_CHANNEL_REF chanDst; /* Channel to update */
nErrcount = IG_load_file("test.jpg", &hImage);
chanSrc.hImage = hImage;
chanSrc.uNumber = 0;
chanDst.hImage = hImage;
chanDst.uNumber = 2;
nErrcount = IG_image_channel_update(&chanSrc, &chanDst);
nErrcount = IG_save_file(hImage, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

Remarks:

Each channel is specified by a HIGEAR image handle and the channel's index within that image. The source and destination channels can reside in different images or the same image.

1.3.1.2.15.9 IG_image_channels_combine

This function creates a new image with the color space specified by colorSpace.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channels_combine(
        LPCAT_CHANNEL_REF channels,
        AT_UINT channelsQty,
        enumIGColorSpaceIDs colorSpace,
        LPHIGEAR hIGear
);
```

Arguments:

Name	Type	Description
channels	LPCAT_CHANNEL_REF	Array of channel descriptors for channels to combine.
channelsQty	AT_UINT	Number of channels to combine.
colorSpace	enumIGColorSpaceIDs	Color space of new image.
hIGear	LPHIGEAR	Pointer to HIGEAR handle where new image is returned.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
/* Make a copy of a 24-bit RGB image in which the
   red and blue channels are swapped */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
HIGEAR hImageCombined;  /* Handle of combined image */
AT_CHANNEL_REF chan[3]; /* Channels to combine */
nErrcount = IG_load_file("test.jpg", &hImage);
chan[0].hImage = hImage;
chan[0].uNumber = 2;
chan[1].hImage = hImage;
chan[1].uNumber = 1;
chan[2].hImage = hImage;
chan[2].uNumber = 0;
nErrcount = IG_image_channels_combine(chan, 3,
    IG_COLOR_SPACE_ID_RGB, &hImageCombined);
nErrcount = IG_save_file(hImageCombined, "test.bmp",
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
IG_image_delete(hImageCombined);
```

Remarks:

The channels in the new image are copied from the channels specified by channels.

This function does not do any color space or pixel conversions, it merely merges the pixel data from different channels together.

1.3.1.2.15.10 IG_image_channels_separate

This function separates channels in an image by creating a new image for each channel.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_channels_separate(
        HIGEAR hIGear,
        LPAT_CHANNEL_REF channels,
        AT_UINT channelsQty
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image in which to store separated channels.
channels	LPAT_CHANNEL_REF	Array of channel descriptors for channels to separate.
channelsQty	AT_UINT	Number of channels to separate.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
/* Load an image and save all of its channels as
   separate pages in a multi-page TIFF file */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of source image */
AT_CHANNEL_REF *chan;   /* Channels to combine */
AT_INT nChannels;       /* Number of channels in image */
AT_INT i;
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
chan = (AT_CHANNEL_REF *) malloc(nChannels * sizeof(AT_CHANNEL_REF));
for (i = 0; i < nChannels; i++)
{
    chan[i].hImage = hImage;
    chan[i].uNumber = i;
}
nErrcount = IG_image_channels_separate(hImage, chan, nChannels);
for (i = 0; i < nChannels; i++)
{
    nErrcount = IG_fltr_save_file(chan[i].hImage, "test.tif",
        IG_SAVE_TIF_UNCOMP, i, !i);
    IG_image_delete(chan[i].hImage);
}
IG_image_delete(hImage);
free(chan);
```

Remarks:

Each new image contains one channel with creates a new image for each channels array of pages, where each page contains one channel with color space set to grayscale. This function does not do any color space or pixel conversions.

1.3.1.2.16 Image DIB Functions

This section provides information about the Image DIB group of functions.

- IG_image_DIB_export
- IG_image_DIB_export_size_calc
- IG_image_DIB_import
- IG_image_DIB_info_get
- IG_image_DIB_palette_pntr_get
- IG_image_DIB_raster_pntr_get

1.3.1.2.16.1 IG_image_DIB_export

This function exports the contents of hIGear into a buffer, provided by the application, using Windows DIB or ImageGear AT_DIB format.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_export(
   const HIGEAR hIGear,
   AT_VOID* lpBuffer,
   AT_INT BufferSize,
   const AT_DIB_EXPORT_OPTIONS* lpOptions
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle of image from which to export DIB.
lpBuffer	AT_VOID*	Memory buffer where DIB will be exported.
BufferSize	AT_INT	Size of memory buffer - use IG_image_DIB_export_size_calc to determine what to use for this.
lpOptions	const AT_DIB_EXPORT_OPTIONS*	Export options.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1...8 bpp;
- Grayscale - 8...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
AT_INT nDibSize;        // Exported DIB size
AT_DIB_EXPORT_OPTIONS Options; // Options for DIB export
LPAT_DIB lpDIBBuffer;    // Buffer to export DIB

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get exported DIB size, allocate memory buffer and export DIB
    memset(&Options, 0, sizeof(AT_DIB_EXPORT_OPTIONS));
    Options.Format = IG_DIB_EXPORT_FORMAT_IG_LEGACY;
    Options.UseAlpha = FALSE;
    IG_image_DIB_export_size_calc(hIGear, &nDibSize, &Options);
    lpDIBBuffer = (LPAT_DIB)malloc(nDibSize);
    nErrcount = IG_image_DIB_export(hIGear, lpDIBBuffer, nDibSize, &Options);
    // ...
    // Delete memory
    free(lpDIBBuffer);
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

hIGear remains valid after calling this function.

1.3.1.2.16.2 IG_image_DIB_export_size_calc

This function calculates the size of the exported DIB for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_export_size_calc(
   const HIGEAR hIGear,
   AT_INT* lpDIBSize,
   const AT_DIB_EXPORT_OPTIONS* lpOptions
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle of image for which to calculate exported DIB size.
lpDIBSize	AT_INT*	Pointer to where exported DIB size will be stored.
lpOptions	const AT_DIB_EXPORT_OPTIONS*	Export options. See IG_image_DIB_export for details.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If lpOptions->Format is IG_DIB_EXPORT_FORMAT_WINDOWS:

- Indexed RGB - 1, 4, 8 bpp
- Grayscale - 8 bpp
- RGB - 24 bpp

If lpOptions->Format is IG_DIB_EXPORT_FORMAT_IG_LEGACY:

- Indexed RGB - 1, 4, 8 bpp
- Grayscale - 8...16 bpp
- RGB - 24 bpp
- CMYK - 32 bpp

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
AT_INT nDibSize;        // Exported DIB size
AT_DIB_EXPORT_OPTIONS Options; // Options for DIB export

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get exported DIB size, allocate memory buffer and export DIB
    memset(&Options, 0, sizeof(AT_DIB_EXPORT_OPTIONS));
    Options.Format = IG_DIB_EXPORT_FORMAT_IG_LEGACY;
    Options.UseAlpha = FALSE;
    IG_image_DIB_export_size_calc(hIGear, &nDibSize, &Options);
    // ...
}
// Destroy the image
IG_image_delete(hIGear);
```

Remarks:

Use this function to calculate the minimal buffer size required to export a DIB with IG_image_DIB_export.

1.3.1.2.16.3 IG_image_DIB_import

This function creates a new HIGEAR image from a Windows DIB stored in memory.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_import(
   const AT_DIB* lpDIB,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpDIB	const AT_DIB*	DIB to be imported.
lphIGear	LPHIGEAR	HIGEAR handle of image created from imported DIB.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp
- Grayscale - 8...16 bpp
- RGB - 24 bpp
- CMYK - 32 bpp

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
HIGEAR hIGearImported;  // Handle of the imported image
AT_INT nDibSize;        // Exported DIB size
AT_DIB_EXPORT_OPTIONS Options; // Options for DIB export
LPAT_DIB lpDIBBuffer;    // Buffer to export DIB

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);

if(nErrcount == 0)
{
    // Get exported DIB size, allocate memory buffer and export DIB
    memset(&Options, 0, sizeof(AT_DIB_EXPORT_OPTIONS));
    Options.Format = IG_DIB_EXPORT_FORMAT_WINDOWS;
    Options.UseAlpha = FALSE;
    IG_image_DIB_export_size_calc(hIGear, &nDibSize, &Options);
    lpDIBBuffer = (LPAT_DIB)malloc(nDibSize);
    nErrcount = IG_image_DIB_export(hIGear, lpDIBBuffer, nDibSize, &Options);
    if(nErrcount == 0)
    {
        // Import the DIB into the new image
        nErrcount = IG_image_DIB_import(lpDIBBuffer, &hIGearImported);
        //...
        // Destroy the image
        IG_image_delete(hIGearImported);
    }
    // Delete memory
    free(lpDIBBuffer);
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

This function copies pixels to the new HIGEAR. Application continues to own lpDIB after calling this function.

1.3.1.2.16.4 IG_image_DIB_info_get

This function returns the handle of a new DIB info object containing information about the given image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_info_get(
   HIGEAR hIGear,
   HIGDIBINFO* lphDIB
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image from which to create DIB info object.
lphDIB	HIGDIBINFO*	Returned DIB info object handle.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
HIGDIBINFO hDIBInfo;    // DIB info handle

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_image_DIB_info_get(hIGear, &hDIBInfo);
    // ...
    // Delete DIBInfo object
    IG_DIB_info_delete(hDIBInfo);
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The application must delete the HIGDIBINFO object after it is done using it by calling IG_DIB_info_delete.

1.3.1.2.16.5 IG_image_DIB_palette_pntr_get

This function returns the address of the image's DIB palette.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_palette_pntr_get(
   HIGEAR hIGear,
   LPAT_RGBQUAD FAR* lpRGBQ,
   LPUINT lpEntries
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRGBQ	LPAT_RGBQUAD FAR*	Pointer to a variable of type LPAT_RGBQUAD, in which this function will store the address of the start of the image's DIB palette.
lpEntries	LPUINT	Pointer to a variable in which will be returned the number of entries in the palette.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
LPAT_RGBQUAD palette = NULL; // Palette pointer
UINT entries = 0;            // Number of entries in the palette

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_image_DIB_palette_pntr_get(hIGear, &palette, &entries);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

HIGEAR DIB palette is stored as an array of AT_RGBQUAD structs. This function returns the address of the first AT_RGBQUAD struct in this array.

If the image doesn't have a palette, the function returns NULL in lpRGBQ.

When referencing a DIB palette, remember that each DIB palette entry contains 4 bytes, not 3, and that the order of the bytes is: Blue, Green, Red, Unused (not Red, Green, Blue).

1.3.1.2.16.6 IG_image_DIB_raster_pntr_get

This function gets a pointer to the beginning of pixel data for a raster in the given image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_DIB_raster_pntr_get(
   HIGEAR hIGear,
   AT_PIXPOS row,
   AT_VOID** lplpRaster
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
row	AT_PIXPOS	Row in image for which to retrieve raster pointer.
lplpRaster	AT_VOID**	Returned pointer to the 1st pixel in the raster.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  // Number of errors on stack
HIGEAR hIGear;          // Handle of image
LPAT_VOID lpRst;        // Pointer to a raster

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_image_DIB_raster_pntr_get(hIGear, 0, &lpRst);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

Rasters are stored from top to bottom, are DWORD-padded on 32-bit platforms (QWORD-padded on 64-bit platforms), use 8, 16, or 32 bits to store channel values, and have alpha/extra channels included in-line with the color channels. For example, an RGB image with an alpha channel would be stored as: RGBA RGBA RGBA ...

If the image is 1bpp b/w, it is stored in a compressed run ends format. See IG_runs_row_get/IG_runs_row_set for a description of this format. It is recommended to use these functions to access run ends data, but you can use IG_image_DIB_raster_pntr_get() with the following restrictions:

- You can only read data. It is not safe to write data.
- You can only access the raster to which you've retrieved a pointer.

Note:

Although ImageGear currently stores uncompressed (non-run ends) rasters continously, we do not recommend that you rely on this, as the internal storage format may change in the future. Use IG_image_DIB_raster_pntr_get to access each raster separately.

1.3.1.2.17 Image Colorspace Functions

This section provides information about the Image Colorspace group of functions.

- IG_image_colorspace_convert
- IG_image_colorspace_get

1.3.1.2.17.1 IG_image_colorspace_convert

This function converts an image to the specified color space.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_colorspace_convert(
        HIGEAR hIGear,
        enumIGColorSpaceIDs newColorSpace,
        LPCAT_COLORSPACE_CONVERSION_OPTIONS options
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
newColorSpace	enumIGColorSpaceIDs	Color space to which to convert.
options	LPCAT_COLORSPACE_CONVERSION_OPTIONS	Conversion options (or NULL).

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* HIGEAR handle of image */
IG_image_colorspace_convert(hImage, IG_COLOR_SPACE_ID_RGB, NULL);
```

Remarks:

Specify the new color space using a value from enumIGColorSpaceIDs. Argument options control the conversion flow. You may pass NULL for options if you don't want to specify any options.

1.3.1.2.17.2 IG_image_colorspace_get

This function gets an image's color space ID.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_image_colorspace_get(
        HIGEAR hIGear,
        enumIGColorSpaceIDs* lpColorspace
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpColorspace	enumIGColorSpaceIDs*	Returned color space of the image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
enumIGColorSpaceIDs cs; /* Color space ID */
AT_BOOL bIndexed;       /* Is the image indexed? */
nErrcount = IG_image_colorspace_get(hImage, &cs);
if ((cs & IG_COLOR_SPACE_ID_ColorMask) == IG_COLOR_SPACE_ID_I)
    bIndexed = TRUE;
else
    bIndexed = FALSE;
```

Remarks:

An ImageGear color space ID is actually a combination of values. It contains information about 1) color channels, 2) an alpha channel, and 3) extra channels. Examine the definition of enumIGColorSpaceIDs carefully (in accucnst.h) and use bitmasks such as IG_COLOR_SPACE_ID_ColorMask to isolate the information you want.

1.3.1.2.18 Image Processing Functions

This section provides information about the Image Processing group of functions.

- IG_IP_add_tilt
- IG_IP_alpha_create
- IG_IP_area_info_get
- IG_IP_arithmetic
- IG_IP_arithmetic_rect
- IG_IP_blend_percent
- IG_IP_blend_with_LUT
- IG_IP_color_combine_ex
- IG_IP_color_convert
- IG_IP_color_count_get
- IG_IP_color_promote
- IG_IP_color_reduce_bayer
- IG_IP_color_reduce_diffuse
- IG_IP_color_reduce_halftone
- IG_IP_color_reduce_median_cut
- IG_IP_color_reduce_octree
- IG_IP_color_reduce_popularity
- IG_IP_color_reduce_to_bitonal
- IG_IP_color_separate
- IG_IP_contrast_adjust
- IG_IP_contrast_adjust_ex
- IG_IP_contrast_equalize
- IG_IP_contrast_gamma
- IG_IP_contrast_invert
- IG_IP_contrast_stretch
- IG_IP_convert_to_gray
- IG_IP_convolve_matrix
- IG_IP_crop
- IG_IP_decrypt
- IG_IP_deskew_angle_find
- IG_IP_deskew_auto
- IG_IP_despeckle
- IG_IP_draw_frame
- IG_IP_drop_shadow
- IG_IP_edge_detection
- IG_IP_edge_map
- IG_IP_encrypt
- IG_IP_enhance_local
- IG_IP_find_tilt
- IG_IP_flip
- IG_IP_gaussian_blur
- IG_IP_geom_despeckle
- IG_IP_histo_clear
- IG_IP_histo_tabulate
- IG_IP_maximum
- IG_IP_median
- IG_IP_merge
- IG_IP_minimum
- IG_IP_NR_ROI_control_get
- IG_IP_NR_ROI_control_set
- IG_IP_NR_ROI_mask_associate
- IG_IP_NR_ROI_mask_delete

- IG_IP_NR_ROI_mask_unassociate
- IG_IP_NR_ROI_to_HIGEAR_mask
- IG_IP_pseudocolor_limits
- IG_IP_pseudocolor_small_grads
- IG_IP_remove_tilt
- IG_IP_resize
- IG_IP_resize_bkgrnd
- IG_IP_resize_bkgrnd_ex
- IG_IP_resize_canvas
- IG_IP_resize_ex
- IG_IP_RGB_to_hue
- IG_IP_rotate_any_angle
- IG_IP_rotate_any_angle_bkgrnd
- IG_IP_rotate_any_angle_ex
- IG_IP_rotate_compute_size
- IG_IP_rotate_multiple_90
- IG_IP_rotate_multiple_90_opt
- IG_IP_sharpen
- IG_IP_smooth
- IG_IP_swap_red_blue
- IG_IP_thumbnail_create
- IG_IP_thumbnail_create_ex
- IG_IP_transform_with_LUT
- IG_IP_transform_with_LUT_ex
- IG_IP_unsharp_mask

1.3.1.2.18.1 IG_IP_add_tilt

This function adds a specified plane to the input image to correct for a tilt in the image luminance.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_add_tilt(
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        AT_DOUBLE dSlopeX,
        AT_DOUBLE dSlopeY
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to be processed.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image on which to operate. If NULL, this operation will be performed on the entire image. Before ImageGear performs this operation it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
dSlopeX	AT_DOUBLE	Specify the slope of the plane in X direction, and must be given in units per pixel.
dSlopeY	AT_DOUBLE	Specify the slope of the plane in X direction, and must be given in units per pixel.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Grayscale - 8, 16, 32 bpp.

Example:

```
HIGEAR    hIGear;         /* HIGEAR handle of image */
AT_RECT lpRect;          /* rectangle to process */
AT_DOUBLE dSlopeX, dSlopeY;
...
IG_IP_add_tilt( hIGear, lpRect, dSlopeX, dSlopeY);
...
```

Remarks:

The plane has X and Y zero crossing at the center of the input image with dSlopeX and dSlopeY as specified in the arguments.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.2 IG_IP_alpha_create

This function has been deprecated and will be removed from the public API in a future release. Please use IG_image_create instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_alpha_create(
        HIGEAR hIGear,
        HIGEAR hIBackgrnd,
        AT_MODE nCreateMode,
        LPHIGEAR lphAlpha
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
hIBackgrnd	HIGEAR	HIGEAR handle to a background image.
nCreateMode	AT_MODE	An integer value of type AT_MODE that tells ImageGear what bit depth the alpha channel should have. The possible settings for this variable, which are defined in accucnst.h are: IG_ALPHA_CREATE_1 and IG_ALPHA_CREATE_8.
lphAlpha	LPHIGEAR	Far pointer to the newly created alpha channel.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

This function creates an alpha channel.

The difference from function IG_image_create_alpha() is that newly created alpha channel is not associated with any HIGEAR - it's only created and returned via last parameter lphAlpha. Other parameters have the same meaning as those of IG_image_create_alpha().

See also the section ImageGear Alpha Channel Support.

1.3.1.2.18.3 IG_IP_area_info_get

This function provides information about a rectangular area of the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_area_info_get(
   HIGEAR hIGear,
   const LPAT_RECT lpRect,
   HIGPIXEL lpPixel,
   AT_MODE nChannel,
   AT_MODE nInfo
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Handle of the image for which to obtain the information.
lpRect	const LPAT_RECT	Rectangle to get the information from (pass NULL for getting information on the whole image).
lpPixel	HIGPIXEL	Receives the calculated value. Depending on the "DIB.PIX_ACCESS_USE_LEGACY_MODE" global control parameter - either a HIGPIXEL object handle, or a pointer to an array of AT_PIXEL. See Remarks.
nChannel	AT_MODE	Specifies a channel or a channel range to get the info about. See enumIGColorChannels for possible values.
nInfo	AT_MODE	Specifies the kind of information to be obtained. See enumDIBAreaInfo for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func
AT_INT channelCount;      // Count of channels in the image
AT_INT bitsPerChannel;      // Channel depth
HIGPIXEL hPixel;

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Set IG_PIX_ACCESS_MODE_NEW access mode
    AT_LMODE AccessMode = IG_PIX_ACCESS_MODE_NEW;
    IG_gctrl_item_set("DIB.PIX_ACCESS_USE_LEGACY_MODE", AM_TID_AT_LMODE, &AccessMode,
sizeof(AT_LMODE), NULL);

    IG_image_channel_count_get(hIGear, &channelCount);
    IG_image_channel_depth_get(hIGear, 0, &bitsPerChannel);

    hPixel = IG_pixel_create(channelCount, bitsPerChannel);
    if(hPixel != NULL)
    {
        // Get average pixel value
        nErrcount = IG_IP_area_info_get(hIGear, NULL, hPixel, IG_COLOR_COMP_ALL,
IG_DIB_AREA_INFO_AVE);
        // ...
        // Delete pixel
        IG_pixel_delete(hPixel);
    }
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

Behavior of this function depends on the "DIB.PIX_ACCESS_USE_LEGACY_MODE" global control parameter.

If "DIB.PIX_ACCESS_USE_LEGACY_MODE" is set to IG_PIX_ACCESS_MODE_NEW, the function expects that lpPixel is set to an HIGPIXEL object handle. The calculated value will be returned in this object. Use IG_pixel_create to create an HIGPIXEL object, specifying the correct number of channels and number of bits per channel. Use IG_pixel_delete to delete it when it is no longer in use.

If the global parameter is set to IG_PIX_ACCESS_MODE_LEGACY, the function expects that lpPixel is a pointer to an array of AT_PIXEL. Length of the array must correspond to the number of image channels. The area info will be scaled to the AT_PIXEL range (BYTE). For example, if the image has 16 bits per channel, and the area info corresponds to 32767, the return value will be (32767 / 256) = 128.

See Pixel Access Modes section for more information on pixel access modes.

Default value of "DIB.PIX_ACCESS_USE_LEGACY_MODE" global control parameter is IG_PIX_ACCESS_MODE_LEGACY.

1.3.1.2.18.4 IG_IP_arithmetic

This function performs an arithmetic or logical operation on two images.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_arithmetic(
   HIGEAR hIGear1,
   HIGEAR hIGear2,
   AT_MODE nOperation
);
```

Arguments:

Name	Type	Description
hIGear1	HIGEAR	HIGEAR handle of image 1, also the destination of the operation.
hIGear2	HIGEAR	HIGEAR handle of image 2.
nOperation	AT_MODE	Specifies the kind of arithmetic operation to perform on the images. See enumIGMergeModes for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear1;              // HIGEAR handle of the first image
HIGEAR hIGear2;              // HIGEAR handle of the second image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear1);
if(nErrcount == 0)
{
    nErrcount = IG_load_file("picture.tif", &hIGear2);
    if(nErrcount == 0)
    {
        nErrcount = IG_IP_arithmetic(hIGear1, hIGear2, IG_ARITH_ADD);
        // ...
        // Destroy the second image
        IG_image_delete(hIGear2);
    }
    // Destroy the first image
    IG_image_delete(hIGear1);
}
```

Remarks:

IG_IP_arithmetic_rect is an extended version of IG_IP_arithmetic that allows specifying a rectangular area on the first image to be used for processing.

IG_IP_merge is an extended version of IG_IP_arithmetic that allows specifying a rectangular area on the first image to be used for processing, as well as the coordinates in the first image where to place the upper-left corner of the specified rectangle of the second image.

1.3.1.2.18.5 IG_IP_arithmetic_rect

This function performs an arithmetic or logical operation on two images, allowing you to specify a rectangular region of the first image on which to perform the operation.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_arithmetic_rect(
   HIGEAR hIGear1,
   HIGEAR hIGear2,
   LPAT_RECT lpImageRect2,
   AT_MODE nOperation
);
```

Arguments:

Name	Type	Description
hIGear1	HIGEAR	HIGEAR handle of image 1, which is also the destination image.
hIGear2	HIGEAR	HIGEAR handle of image 2.
lpImageRect2	LPAT_RECT	Specifies a rectangle in hIGear2, which will be used for processing. Set to NULL for the whole image.
nOperation	AT_MODE	Specifies the kind of arithmetic operation to perform on the images. See enumIGMergeModes for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear1;              // HIGEAR handle of the first image
HIGEAR hIGear2;              // HIGEAR handle of the second image
AT_DIMENSION nWidth, nHeight;  // Dimensions of the first image
AT_ERRCOUNT nErrcount;     // Count of errs on stack upon ret from func
AT_RECT rcRect;              // Region to merge

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear1);
if(nErrcount == 0)
{
    nErrcount = IG_load_file("picture.tif", &hIGear2);
    if(nErrcount == 0)
    {
        // Get dimensions of the first image and initialize merging region
        IG_image_dimensions_get(hIGear1, &nWidth, &nHeight, NULL);
        rcRect.left = 0;
        rcRect.top = 0;
        rcRect.right = nWidth / 2;
        rcRect.bottom = nHeight / 2;

        nErrcount = IG_IP_arithmetic_rect(hIGear1, hIGear2, &rcRect, IG_ARITH_ADD);
        // ...
        // Destroy the second image
        IG_image_delete(hIGear2);
    }
    // Destroy the first image
    IG_image_delete(hIGear1);
}
```

Remarks:

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can merge a rectangular sub-region of hIGear2 into hIGear1. However, before ImageGear performs the operation specified by this function, it will check to see if an internal NRA flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask.

IG_IP_merge is an extended version of IG_IP_arithmetic_rect that allows specifying a rectangular area on the first image to be used for processing, as well as the coordinates in the first image where to place the upper-left corner of the specified rectangle of the second image.

1.3.1.2.18.6 IG_IP_blend_percent

This function blends the second image into the first.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_blend_percent (
        HIGEAR hIGearDest,
        const HIGEAR hIGear2,
        const DOUBLE dblPctOfImage2,
        const AT_MODE nColorChannel,
        const LPAT_RECT lpRect
);
```

Arguments:

Name	Type	Description
hIGearDest	HIGEAR	HIGEAR handle of the image into which to be blended.
hIGear2	const HIGEAR	HIGEAR handle of image to blend in, must be same size and bit depth.
dblPctOfImage2	const DOUBLE	Percent of image 2 to be in the blend (the percent of image 1 will be 100.0 minus this). 0 means all of image 1; 50 means half and half, and 100 means all of image 2. The range of values is 0.0 to 100.0.
nColorChannel	const AT_MODE	A constant such as IG_COLOR_COMP_R, _G, _B, or _RGB, specifying which color(s) to blend.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate on. lpRect = NULL means the entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non- rectangular ROI defined by the mask.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear1;       /* Handle of image1 that will change  */
HIGEAR hIGear2;      /* Handle of image2      */
AT_ERRCOUNT nErrcount;  /* Returned count of errors  */
AT_DIMENSION nWidth, nHeight;  /* Width & Height of Image1          */
UINT nBpp;     /* Bits per pixel    */
AT_RECT rcRectOf1;    /* selected rectangle from image1  */
/* Get dimensions of hIGear1, so can use width and height values below */
nErrcount = IG_image_dimensions_get( hIGear1, &nWidth, &nHeight, &nBpp );
/*Use the bottom ? of Image1 */
rcRectOf1.left = 0;
rcRectOf1.top = nHeight/2;
rcRectOf1.right = nWidth - 1;
rcRectOf1.bottom = nHeight - 1;
/* If 24-bit image, blend all color channels (else blend pixels) */
nErrcount = IG_IP_blend_percent ( hIGear1, hIGear2, 20.0,IG_COLOR_COMP_RGB, &rcRectOf1 );
```

Remarks:

Use lprect to set a rectangular portion of the first image to be processed. dblPctOfImage2 specifies the percent of each image to be used in the result. The image in hIGearDest is destroyed and is replaced with the resulting blend. The percentage ranges from 0 to 100. A zero results in 0% hIGearDest and 100% hIGear2. A value of 50% results in an image which is created ? of each image. A value of 100% creates an image that is 100% hIGear2 and 0% hIGearDest. The images must be the same width, height, and bit depth. nColorChannel lets you specify that only one color of a 24-bit image is to be blended in.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() for more details.

1.3.1.2.18.7 IG_IP_blend_with_LUT

This function blends two images using Look-Up Tables (LUTs) to determine the strength of each pixel's contribution.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_blend_with_LUT (
        HIGEAR hIGearDest,
        HIGEAR hIGear2,
        const LPAT_LUT lpLUT_red,
        const LPAT_LUT lpLUT_green,
        LPAT_LUT lpLUT_blue,
        const LPAT_RECT lpRect
);
```

Arguments:

Name	Type	Description
hIGearDest	HIGEAR	HIGEAR handle of the image into which to be blended.
hIGear2	HIGEAR	HIGEAR handle of image to blend; must be same size and bit depth.
lpLUT_red	const LPAT_LUT	Far pointer to Red channel of LUT for RGB images; also used as a single pointer to LUT for grayscale images.
lpLUT_green	const LPAT_LUT	Far pointer to Green channel of LUT for RGB image; not used for grayscale images.
lpLUT_blue	LPAT_LUT	Far pointer to Blue channel of LUT of RGB image; not used for grayscale images.
lpRect	const LPAT_RECT	Rectangular portion of the image to process. Set to NULL for the whole image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGearDest    /* HIGEAR handle of image 1, destination */
                hIGear2;   /* HIGEAR handle of image 2    */
AT_LUT LUT_blend[256];   /* The LUT for blend values */
INT pix;            /* Loop index, = pixel value  */
AT_ERRCOUNT nErrcount;     /* Returned count of errors   */
for ( pix = 0;  pix < 256;  pix++ )
        { LUTblend[pix]  =  25; }     /* set Look-Up Table   */
for (pix = 10; pix < 100; pix++)
        { LUTblend[pix] = 75; }
nErrcount =  IG_IP_blend_with_LUT ( hIGearDest, hIGear2, (LPAT_LUT)&LUT_blend,
        (LPAT_LUT)&LUT_blend, (LPAT_LUT)&LUT_blend, NULL );
```

Remarks:

The pixel values from hIGearDest are used as the indexes into the LUTs. Both hIGear2 and hIGearDest must be the same bit depth and dimensions. The image in hIGearDest is destroyed and is replaced with the resulting blend.

For RGB images, each channel (R, G, or B) of the hIGearDest image is processed through its own LUT.

For indexed and grayscale images, the lpLUT_red is used and the other two are ignored (you can pass in NULL for these).

Images that have other colorspaces, such as CMYK or LAB, are converted into RGB for processing internally, and then converted back to their original colorspace. The function works on these images as if they had RGB colorspace.

The function does not process Alpha and Extra channels, if they are present in the image.

Each LUT that is to be used must point to a LUT that has at least enough entries to process the images being passed in. For images having up to 8 bits per channel, LUTs must contain 256 bytes. For images having up to 16 bits per channel, LUT must contain 65536 bytes.

Each entry into the LUT determines the percentage of the blend on a pixel-by-pixel basis. Pixel values of the hIGearDest image are used as indexes into the array. The LUTs should be initialized with values from 0 to 100, where:

- 0 = 0 % hIGearDest, 100% hIGear2
- 50 = 50% of hIGearDest and 50% of hIGear2
- 100 = 100% hIGearDest and 0 % hIGear2

For grayscale images, the value of each pixel in hIGearDest is used as the index into the LUT. For RGB images, it is the intensity, calculated as (R+G+B)/3 that is used. For other colorspaces, pixels are converted to RGB and then intensity is calculated.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. (See above.) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.8 IG_IP_color_combine_ex

This function is an upgrade to IG_IP_color_combine().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_combine_ex (
        LPHIGEAR lphIGear_result,
        HIGEAR hIGear1,
        HIGEAR hIGear2,
        HIGEAR hIGear3,
        HIGEAR hIGear4,
        AT_MODE color_space,
        AT_MODE dst_color_space
);
```

Arguments:

Name	Type	Description
lphIGear_result	LPHIGEAR	Far pointer to an object of type HIGEAR, to receive the HIGEAR handle of the created 24- bit image.
hIGear1	HIGEAR	HIGEAR handle of input image to be treated as channel 1.
hIGear2	HIGEAR	HIGEAR handle of input image to be treated as channel 2.
hIGear3	HIGEAR	HIGEAR handle of input image to be treated as channel 3.
hIGear4	HIGEAR	HIGEAR handle of input image to be treated as channel 4.
nColorSpace	AT_MODE	A constant such as IG_COLOR_SPACE_RGB describing how the channels are to be merged.
dst_color_space	AT_MODE	tells what color space output HIGEAR should have. There are 2 possible values now: RGB and CMYK.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

While support for IG_IP_color_combine() is being maintained, it is recommended that you use this newer function. The additional benefit of this function is that it allows you to choose a color space for the destination image. For example, you can choose to store your destination image using the CMYK color scheme. Currently, the dst_color_space parameter can take one of the following constants as its setting: IG_COLOR_SPACE_RGBand IG_COLOR_SPACE_CMYK. In order to set it to CMYK, you must be sure that you first call IG_color_space_level_set() function with settings of IG_COLOR_SPACE_CMYK and IG_FULL_SUPPORT.

1.3.1.2.18.9 IG_IP_color_convert

This function has been deprecated and will be removed from the public API in a future release. Please use IG_image_colorspace_convert instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_convert(
        [IN] HIGEAR hIGear,
        [IN] AT_MODE nColorSpace
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	The ImageGear handle of an image.
nColorSpace	AT_MODE	The identifier of destination color space where to convert.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image  */
. . .
IG_IP_color_convert( hIGear, IG_COLOR_SPACE_CMYK );
. . .
```

Remarks:

This function converts an image from one internal format to another depending on the parameter, nColorSpace.

The nColorSpace parameter must be a color space listed in enumIGColorSpaces, which is defined in accucnst.h.

Commonly used values for nColorSpace are:

- IG_COLOR_SPACE_CMYK
- IG_COLOR_SPACE_RGB

1.3.1.2.18.10 IG_IP_color_count_get

This function counts the number of unique colors in the specified rectangular area of the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_count_get(
        HIGEAR hIGear,
        LPAT_RECT lpRect,
        LPAT_INT lpCount
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image whose count colors are to be counted.
lpRect	LPAT_RECT	Rectangle in which to count colors (NULL for the whole image).
lpCount	LPAT_INT	Pointer to AT_INT to receive number of colors.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

1.3.1.2.18.11 IG_IP_color_promote

This function promotes an image to the common pixel formats of 4-bit Indexed, 8-bit Indexed, 24-bit RGB, or 32-bit CMYK.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_promote(
   HIGEAR hIGear,
   AT_MODE nPromoteTo
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nPromoteTo	AT_MODE	Specifies the depth to which to promote. See enumIGPromotionModes for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

For Indexed images, bit depth should be no less than the depth specified by the nPromoteTo parameter.

Example:

```
HIGEAR hIGear;             // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;     // Count of errs on stack upon ret from func

// Load image file "picture.tif", 1 bpp, from working directory
nErrcount = IG_load_file("picture.tif", &hIGear);
if(nErrcount == 0)
{
    // Promote to RGB 24
    nErrcount = IG_IP_color_promote(hIGear, IG_PROMOTE_TO_24);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_image_colorspace_convert and IG_image_channel_depths_change instead.

1.3.1.2.18.12 IG_IP_color_reduce_bayer

This function reduces the image to a fewer number of Bits Per Pixel, using a Bayer dithering algorithm.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_bayer (
        HIGEAR hIGear,
        UINT nToBits,
        LPAT_RGBQUAD lpPalette
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image of which to reduce bit depth.
nToBits	UINT	Number of bits per pixel after reduction (4 or 1).
lpPalette	LPAT_RGBQUAD	This argument is currently not used.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB – 4, 8 bpp;
Grayscale – 4, 8 bpp;
RGB – 24 bpp.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of 4, 8, or 24 bit image */
/* Reduce image to 1-bit black-and-white: */
IG_IP_color_reduce_bayer ( hIGear, 1, NULL );
```

Remarks:

The target bit depth is specified by argument nToBits. In general, a color image will be reduced to a fewer number of colors, and a grayscale image will be reduced to a fewer number of shades of gray. Note that setting nToBits = 1 will reduce the image to monochrome or black-and-white.

The input number of Bits Per Pixel must be greater than nToBits, or an error will result.

See also the section in entitled Color Reduction.

1.3.1.2.18.13 IG_IP_color_reduce_diffuse

This function reduces the image to a fewer number of Bits Per Pixel.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_diffuse (
        HIGEAR hIGear,
        UINT nToBits,
        INT level,
        LPAT_RGBQUAD lpPalette
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image of which to reduce bit depth.
nToBits	UINT	Number of bits per pixel after reduction, 1 or 4.
Level	INT	Threshold value for dithering, 0 to 255. Has effect only if nToBits = 1.
lpPalette	LPAT_RGBQUAD	This argument is currently not used.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB – 4, 8 bpp;
Grayscale – 4, 8 bpp;
RGB – 24 bpp.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of 4, 8, or 24 bit image */
/* Reduce image to 1-bit black-and-white: */
IG_IP_color_reduce_diffuse ( hIGear, 1, 128, NULL );
```

Remarks:

The target bit depth is specified by argument nToBits. In general, a color image will be reduced to a fewer number of colors, and a grayscale image will be reduced to a fewer number of shades of gray. Note that setting nToBits = 1 will reduce the image to monochrome or black-and-white.

When reducing the image to monochrome (black-and-white), the level parameter sets a threshold value for the target image:

- level = 128 means that black and white in the target image will be in equal proportion.
- level greater than 128 would mean more bright than dark.
- level less than 128 would mean more dark than bright.

The input number of Bits Per Pixel must be greater than nToBits, or an error will result.

See also section in entitled Color Reduction.

1.3.1.2.18.14 IG_IP_color_reduce_halftone

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_halftone (
        HIGEAR hIGear,
        AT_MODE nOption
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nOption	AT_MODE	Halftoning pattern - 0 for now (squares).

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB – 4, 8 bpp;
Grayscale – 4, 8 bpp;
RGB – 24 bpp.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of 4, 8, or 24 bit image */
/* Reduce to a 1-bit halftone image: */
IG_IP_color_reduce_halftone ( hIGear, 0 );
```

Remarks:

This function reduces a color or grayscale image to a 1 bit per pixel image suitable for use in half-toning. The resulting image will consist of small squares of varying sizes that will give the appearance of varying shades of gray.

See also the section in entitled Color Reduction.

1.3.1.2.18.15 IG_IP_color_reduce_median_cut

This function reduces an image by dividing it into nMaxColors equal-sized squares.

Declaration:

```
AT_ERRCOUNT ACCUAPIIG_IP_color_reduce_median_cut (
        HIGEAR hIGear,
        BOOL bFastRemap,
        UINT nMaxColors
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
bFastRemap	BOOL	Set = TRUE for reduction algorithm optimized for speed. Set = FALSE for algorithm optimized for quality.
nMaxColors	UINT	The maximum number of colors (that is, maximum number of unique pixel values) you want in the resulting image. Must be 16 to 256.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

RGB – 24 bpp.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image */
/* Reduce to 64 colors using median cut algorithm */
IG_IP_color_reduce_median_cut ( hIGear, FALSE, 64 );
```

Remarks:

The colors of the pixels in each square will be averaged to produce a resulting color. The resulting image will contain only these colors.

See also section in entitled Color Reduction.

1.3.1.2.18.16 IG_IP_color_reduce_octree

This function reduces a 24-bit or 8-bit image to an 8-bit or 4-bit image, having the number of colors specified by nMaxColors.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_octree (
        HIGEAR hIGear,
        BOOL bFastRemap,
        UINT nMaxColors,
        const UINT nPaletteSize,
        const LPAT_RGB lpPalette
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
bFastRemap	BOOL	Set = TRUE for reduction algorithm optimized for speed. Set = FALSE for algorithm optimized for quality.
nMaxColors	UINT	The maximum number of colors (that is, maximum number of unique pixel values) you want in the resulting image. If you set this to > 16, the resulting image will be an 8-bit. Valid values are 8 - 256. If you pass in a value out of this range, ImageGear will set the value to either 8 or 256. See description below for details.
nPaletteSize	const UINT	Number of entries in palette lpPalette that will be used by ImageGear during reduction. This should be >= nMaxColors. Setting this to 0, and setting lpPalette to NULL will have ImageGear make an optimized palette for you.
lpPalette	const LPAT_RGB	LONG pointer to an array of palette entries, where the number of entries = nPaletteSize. Set to NULL if you want ImageGear to make an optimized palette for you.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB – 8 bpp;
Grayscale – 8 bpp;
RGB – 24 bpp.

Example:

```
HIGEAR hIGear;   /* HIGEAR handle of image   */
AT_RGB rgbPalette[64];  /* Pointer to image's palette */
AT_ERRCOUNT   nErrcount;    /* Tally of ImageGear errors on stack*/
/* Reduce image to 64 colors using given palette */
nErrcount = IG_IP_color_reduce_octree (hIGear, FALSE, 64, 64, rgbPalette);
/* Reduce image to 64 colors, having ImageGear build optimal palette  */
nErrcount = IG_IP_color_reduce_octree (hIGear, FALSE, 64, 0, NULL);
```

Remarks:

If you set nPaletteSize > 0 and supply an address to lpPalette, ImageGear will use your palette. If you set either nPaletteSize = 0 or lpPalette to NULL, ImageGear will build an optimized palette for you. If you set nMaxColors > 16, then an 8-bit image will always result. Setting nMaxColors <=16 will result in a 4-bit image. You may not specify less than 8 colors.

nPaletteSize should be set to >= nMaxColors. If nPaletteSize is set to 0, ImageGear will build an optimal palette and lpPalette will be unused. The table below demonstrates some sample cases of 8 and 24-bit images being reduced, using the setting of nMaxColors (middle column). The right-most column shows the number of Bits Per Pixel that the resulting image will have.

Octree Bit Depths In and Out

Bpp of orig. HIGEAR	**# of resulting colors**	**Bpp reduced image**
24	17-256	8
24	8-16	4
24	1-7	*
8	17-256	8
8	8-16	4
8	1-7	*

*Not possible, nMaxColors will be changed to 8

See also the section in entitled Color Reduction.

1.3.1.2.18.17 IG_IP_color_reduce_popularity

This function reduces a 24-bit image to an 8-bit image while retaining its most popular, or prevalent, colors.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_popularity (
        HIGEAR hIGear,
        BOOL bFastRemap,
        UINT nMaxColors
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
bFastRemap	BOOL	Set = TRUE for reduction algorithm optimized for speed. Set = FALSE for algorithm optimized for quality.
nMaxColors	UINT	Number of colors to which to reduce.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

RGB – 24 bpp.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image */
/* Reduce to 64 colors using octree algorithm   */
IG_IP_color_reduce_popularity ( hIGear, FALSE, 64 );
```

Remarks:

Use nMaxColors to specify the maximum number of colors wanted in the result.

See also the section in entitled Color Reduction.

1.3.1.2.18.18 IG_IP_color_reduce_to_bitonal

This function reduces the hIGear image from 24, 8, or 4 bpp to 1 bpp.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_reduce_to_bitonal(
        HIGEAR hIGear,
        const AT_MODE nOption,
        const UINT nThreshold,
        UINT nWeight1,
        UINT nWeight2,
        UINT nWeight3
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
nOption	const AT_MODE	A constant of type AT_MODE, such as IG_REDUCE_BITONAL_AVE, that specifies what type of reduction to perform. See accucnst.h for the complete list.
nThreshold	const UINT	An integer value that specifies which pixel values will be changed to black, and which pixel values will be changed to white. Pixels with values greater than nThreshold will be changed to white, and pixels that have values less than nThreshold will be changed to black.
nWeight1	UINT	These values are only used if nOption is set to IG_REDUCE_BITONAL_WEIGHTED. The range of values is 0 - 255. The default values are 255.
nWeight2	UINT	The range of values is 0 - 255. The default values are 255.
nWeight3	UINT	The range of values is 0 - 255. The default values are 255.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB – 4, 8 bpp;
Grayscale – 4, 8 bpp;
RGB – 24 bpp.

Example:

```
HIGEAR hIGear,       /* HIGEAR handle of input image   */
AT_ERRCOUNT nErrcount;   /* Returned count of errors   */
nErrcount = IG_IP_color_reduce_to_bitonal ( hIGear, IG_REDUCE_BITONAL_AVE, 100, 0, 0, 0 );
```

Remarks:

Set nOption to specify how to get the threshold value. If you set nOption to IG_REDUCE_BITONAL_WEIGHTED, you may also set the values of nWeight1, nWeight2, and nWeight3. These "weights" are used to determine how much influence the values of the red, green, or blue pixels will have on the reduction. For example, if nWeight1 (red)= 255, nWeight2 (green) = 0, nWeight3 (blue) = 0, the whole reduction will depend on the value of the red pixels. The green and blue pixel values will have "no weight."

IG_REDUCE_BITONAL_GRAYSCALE gives the most weight to the value of green. This optimizes for the perception of the human eye, in which blue is the hardest color to see, and therefore requires the least weight.

IG_REDUCE_BITONAL_AVE gives equal weight to all three pixel values.

Here are the formulas used by the three different reduction methods:

- IG_REDUCE_BITONAL_GRAYSCALE: value = (red*77 + green*151 + blue*28)/256;
- IG_REDUCE_BITONAL_AVE: value = (red + green + blue)/3;
- IG_REDUCE_BITONAL_WEIGHTED: value = (red*w1 + green*w2 + blue*w3)/(w1 + w2 + w3);

Use the nThreshold argument to set the threshold value for converting pixels to black or white. If the value, as calculated by one of the above reduction methods, is less than nThreshold, the pixels will be set to black; if it is greater or equal to nThreshold, the pixels will be set to white.

See also the section in entitled Color Reduction.

1.3.1.2.18.19 IG_IP_color_separate

This function is the reverse of IG_IP_color_combine_ex().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_color_separate (
        HIGEAR hIGearOrig,
        LPHIGEAR lphIGear1,
        LPHIGEAR lphIGear2,
        LPHIGEAR lphIGear3,
        LPHIGEAR lphIGear4,
        AT_MODE nColorSpace
);
```

Arguments:

Name	Type	Description
hIGearOrig	HIGEAR	HIGEAR handle of the original image to be separated.
lphIGear1	LPHIGEAR	Far pointer to an object of type HIGEAR to receive the handle of the separated channel 1 image.
lphIGear2	LPHIGEAR	Far pointer to receive handle of channel 2 image.
lphIGear3	LPHIGEAR	Far pointer to receive handle of channel 3 image.
lphIGear4	LPHIGEAR	Far pointer to receive handle of channel 4 image.
nColorSpace	AT_MODE	A constant such as IG_COLOR_SPACE_RGB or IG_COLOR_SPACE_CMYK describing how the individual channels are to be separated or extracted.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear,     /* HIGEAR handle of input image  */
 hIGearRed, /* Handle of Red output image,    */
 hIGearGreen, hIGearBlue,   /* Green, Blue output images */
 hIGearNULL;    /* (Not used when IG_COLOR_SPACE_RGB)    */
AT_ERRCOUNT nErrcount;  /* Returned count of errors */
nErrcount = IG_IP_color_separate ( hIGear, &hIGearRed, &hIGearGreen, &hIGearBlue,
&hIGearNull, IG_COLOR_SPACE_RGB );
```

Remarks:

This function is the reverse of IG_IP_color_combine_ex(). See the description of that function. Each of the output images created by this function (lphIGear1, etc.) will be grayscale. That is, each will have a grayscale palette. The pixel values will be those obtained from the input image.

1.3.1.2.18.20 IG_IP_contrast_adjust

This function adjusts the brightness and contrast of the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_adjust(
   HIGEAR hIGear,
   LPAT_RECT lpRect,
   AT_MODE nMethodMode,
   DOUBLE dblContrast,
   DOUBLE dblBrightness
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	LPAT_RECT	Specifies a rectangle within the image to operate on. NULL means the entire image. See Remarks below.
nMethodMode	AT_MODE	Specifies whether to alter the pixels or the palette. See enumIGContrastModes.
dblContrast	DOUBLE	Specifies the contrast value. The useful range is from -(2^bpc) to (2^bpc), where bpc is the image bits per channel.
dblBrightness	DOUBLE	Specifies the brightness value. The useful range is from -(2^bpc)+1 to (2^bpc)-1, where bpc is the image bits per channel.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_contrast_adjust(hIGear, NULL, IG_CONTRAST_PIXEL, 2.0, -10.0);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

Brightness and contrast are linear controls that affect the intensity of the image pixels. These controls are similar to the Brightness and Contrast controls on a typical television set.

Contrast is a multiplier, and Brightness is an additive value. The contrast is applied about the middle value of the pixel intensity range. A Contrast of 2.0 will cause each pixel to become twice farther from the middle intensity value, while 0.5 makes each twice closer to it. A Brightness value of 20.0 will cause each pixel's intensity to be increased by 20, and a -20 will decrease or darken each by 20. Pixel values are clipped to the pixel intensity range supported by the image channel depths. Once clipped, the data is lost and cannot be regenerated. A Brightness of 0.0 and a

Contrast of 1.0 will cause no change to the image. A -1.0 Contrast with a Brightness of 0.0 can be used to invert the intensity range.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal NRA flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask.

When IG_CONTRAST_PALETTE is used, the lpRect rectangle is ignored, since the whole image is affected when the palette is changed.

Although the function allows using IG_CONTRAST_PIXEL for indexed images, in most cases such operation will not invert the image, but rather will change image colors in a random looking way, depending on image palette. Only if the palette is linear will adjusting the pixels adjust the display in the desired way. An example of a linear palette is the grayscale palette: R[i] = G[i] = B[i] = i.

IG_IP_contrast_adjust_ex is an extended version of this function that allows adjusting contrast on specific image channels.

1.3.1.2.18.21 IG_IP_contrast_adjust_ex

This function adjusts the brightness and contrast of the specified image channels.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_adjust_ex(
   HIGEAR hIGear,
   LPAT_RECT lpRect,
   AT_MODE nMethodMode,
   DOUBLE dblContrast,
   DOUBLE dblBrightness,
   AT_MODE nColorChannel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	LPAT_RECT	Specifies a rectangle within the image on which to operate. NULL means the entire image. See Remarks below.
nMethodMode	AT_MODE	Specifies whether to alter the pixels or the palette. See enumIGContrastModes.
dblContrast	DOUBLE	Specifies the contrast value. The useful range is from -(2^bpc) to (2^bpc), where bpc is the image bits per channel.
dblBrightness	DOUBLE	Specifies the brightness value. The useful range is from -(2^bpc)+1 to (2^bpc)-1, where bpc is the image bits per channel.
nColorChannel	AT_MODE	Specifies a channel or a channel range to adjust. See enumIGColorChannels for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_contrast_adjust_ex(hIGear, NULL, IG_CONTRAST_PIXEL, 2.0, -10.0,
IG_COLOR_COMP_R);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

Brightness and contrast are linear controls that affect the intensity of the image pixels. These controls are similar to the Brightness and Contrast controls on a typical television set.

Contrast is a multiplier and Brightness is an additive value. The contrast is applied about the middle value of the pixel intensity range. A Contrast of 2.0 will cause each pixel to become twice farther from the middle intensity value, while 0.5 makes each twice closer to it. A Brightness value of 20.0 will cause each pixel's intensity to be increased by 20, and a -20 will decrease or darken each by 20. Pixel values are clipped to the 0 to 255 range. Once clipped, the data is lost and cannot be regenerated. A Brightness of 0.0 and a Contrast of 1.0 will cause no change to the image. A -1.0 Contrast with a Brightness of 0.0 can be used to invert the intensity range.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal NRA flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask.

When IG_CONTRAST_PALETTE is used the lpRect rectangle is ignored, since the whole image is affected when the palette is changed.

> Although the function allows using IG_CONTRAST_PIXEL for indexed images, in most cases such operation will not invert the image, but rather will change image colors in a random looking way, depending on image palette. Only if the palette is linear will adjusting the pixels adjust the display in the desired way. An example of a linear palette is the grayscale palette: R[i] = G[i] = B[i] = i.

1.3.1.2.18.22 IG_IP_contrast_equalize

This function automatically adjusts the contrast of the image so that each range of possible intensities has about the same number of pixels in it.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_equalize(
        HIGEAR hIGear,
        LPAT_RECT lpRect,
        AT_MODE nMethodMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	LPAT_RECT	Far pointer to an AT_RECT struct defining the rectangle within the image that this function is to operate on. If NULL, the entire image will be operated on. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nMethodMode	AT_MODE	IG_CONTRAST_PIXEL or IG_CONTRAST_PALETTE, telling whether to alter the pixels themselves (the image bitmap) or the palette.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;      /* HIGEAR handle of image */
IG_IP_contrast_equalize ( hIGear, NULL, IG_CONTRAST_PIXEL );
```

Remarks:

Just like IG_IP_contrast_stretch(), this will expand the intensity range of the image to fill the entire 0 to 255 range. However, unlike that function this one is non-linear.

This function is often used in x-ray images and in others where the contrast can be very small in the original. IG_IP_contrast_equalize() will bring out subtle changes in the contrast.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.23 IG_IP_contrast_gamma

This function adjusts the contrast of the image using a non-linear gamma method.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_gamma(
   HIGEAR hIGear,
   LPAT_RECT lpRect,
   AT_MODE nMethodMode,
   DOUBLE dblGamma
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	LPAT_RECT	Specifies a rectangle within the image on which to operate. NULL means the entire image. See Remarks below.
nMethodMode	AT_MODE	Specifies whether to alter the pixels or the palette. See enumIGContrastModes.
dblGamma	DOUBLE	Greater than 0.0. Usual range: 0.75 to 3.0.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_contrast_gamma(hIGear, NULL, IG_CONTRAST_PIXEL, 2.0);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

Gamma is a non-linear method to adjust the contrast of a image. In this method, the amount a pixel's intensity changes depends on its original intensity. This can be used to make dark regions brighter without over saturating (clipping) the bright regions. Or, conversely, to make light regions darker without under saturating the dark regions. Gamma was originally introduced to compensate for the non-linear nature of the phosphors used in monitors and in the original tube cameras that created images.

While the gamma can be any non-zero positive value, the usual range is 0.75 to 3.0. A gamma value of 1.0 does not alter the image. For typical monitors, a range of 1.8 to 2.2 is usual. Values less than 1.0 cause dark pixels to become brighter. Values greater than 1.0 cause bright regions to become darker.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal NRA flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask.

1.3.1.2.18.24 IG_IP_contrast_invert

This function inverts every color to its complement within the rectangular portion of the image selected by lpRect.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_invert(
   HIGEAR hIGear,
   LPAT_RECT lpRect,
   AT_MODE nMethodMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image of which to invert contrast.
lpRect	LPAT_RECT	Specifies a rectangle within the image on which to operate. NULL means the entire image. See remarks below.
nMethodMode	AT_MODE	Specifies whether to alter the pixels or the palette. See enumIGContrastModes.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Invert the image
    nErrcount = IG_IP_contrast_invert(hIGear, NULL, IG_CONTRAST_PIXEL);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

For black-and-white images, black will become white, and white will become black. For grayscale and color images, every red, green, and blue color intensity value will be complemented: 0 will become 255 (and vice versa), 1 will become 254, and so on. Therefore, in a grayscale image, the darkest grays will become the lightest grays, and vice versa; and in a color image, colors near green will complement to colors near magenta (the complement of green), and so on.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal NRA flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask.

If nMethodMode = IG_CONTRAST_PIXEL, the inversion is accomplished by inverting all bits of all pixels within lpRect:

bits that are 1 become 0, and bits that are 0 become 1. If nMethodMode = IG_CONTRAST_PALETTE, the inversion is accomplished by inverting the bits in the image's palette (the pixels are left unchanged).

Specifying IG_CONTRAST_PALETTE inverts the entire image, ignoring any rectangle specified.

Although the function allows using IG_CONTRAST_PIXEL for indexed images, in most cases such operation will not invert the image, but rather will change image colors in a random looking way, depending on image palette. Only if the palette is symmetric (R[i] = ^R[^i], G[i] = ^G[^i], G[i] = ^G[^i]) will inverting the pixels actually result in an inverted display. An example of a symmetric palette is the grayscale palette: R[i] = G[i] = B[i] = i.

If the image is not paletted, nMethodMode is ignored.

1.3.1.2.18.25 IG_IP_contrast_stretch

This function automatically adjusts the contrast of the image so that at least one pixel is completely black and one pixel is completely white.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_contrast_stretch (
        HIGEAR hIGear,
        LPAT_RECT lpRect,
        AT_MODE nMethodMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	LPAT_RECT	Far pointer to an AT_RECT struct defining the rectangle within the image that this function is to operate on. If NULL, the entire image will be operated on. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nMethodMode	AT_MODE	IG_CONTRAST_PIXEL or IG_CONTRAST_PALETTE, telling whether to alter the pixels themselves (the image bitmap) or the palette.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */ IG_IP_contrast_stretch ( hIGear,
NULL,IG_CONTRAST_PIXEL );
```

Remarks:

This fills the entire range of the pixel intensities. The original pixel intensities are adjusted linearly between the 2 extremes. (If the image already fills the entire range then the image is not altered.)

Images that use the entire range often appear richer and the colors display more vivid.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. (See above.) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.26 IG_IP_convert_to_gray

This function converts the image referenced by hIGear to a grayscale image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_convert_to_gray( HIGEAR hIGear);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_convert_to_gray(hIGear);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The function changes image color space to IG_COLOR_SPACE_ID_Gy. The bit depth of the resulting image will be equal to the maximal channel depth of the source image. If the original image has an Alpha or Pre-multiplied Alpha channel, the image will be blended over a black background to produce the resulting image. If the original image has Extra channels, they will be removed.

1.3.1.2.18.27 IG_IP_convolve_matrix

This function convolves the image using a user-defined convolution kernel.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_convolve_matrix(
   HIGEAR hIGear,
   LPAT_RECT lpRect,
   LPAT_INT lpMatrix,
   UINT nMatrixWidth,
   UINT nMatrixHeight,
   DOUBLE dblNormalizer,
   AT_MODE nColorChannel,
   AT_MODE nResultForm,
   AT_BOOL bAddToOrigin
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to be processed.
lpRect	LPAT_RECT	Rectangle of image to process; setting to NULL will process the whole image.
lpMatrix	LPAT_INT	Pointer to the array of convolution kernel elements.
nMatrixWidth	UINT	Width of the convolution kernel.
nMatrixHeight	UINT	Height of the convolution kernel.
dblNormalizer	DOUBLE	Normalizer of the convolution kernel.
nColorChannel	AT_MODE	Specifies the color channel or group of channels to be processed. See enumIGColorChannels for possible values.
nResultForm	AT_MODE	Specifies how the result value should be stored. See enumIGConvolutionResults for possible values.
bAddToOrigin	AT_BOOL	Tells whether to add the result of the convolution to the pixel values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;             // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;     // Count of errs on stack upon ret from func

AT_INT          mxConv[5 * 3] =   // Convolution kernel
{
    1,1,1,1,1,
    -2,-2,-2,-2,-2,
    1,1,1,1,1
};

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
```

```
{
    nErrcount = IG_IP_convolve_matrix(hIGear, NULL, mxConv, 5, 3, 1.0,
        IG_COLOR_COMP_RGB, IG_CONV_RESULT_RAW, FALSE);

    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The result of the convolution is multiplied by the normalizer, dblNormalizer. For kernels that sum to zero, the normalizer is usually set to 1.0. When the sum is not zero, the normalizer's value will depend on the goal of convolution. In a non-weighted averaging convolution the kernel elements are often all ones. In this case the normalizer would be equal to 1/(sum of kernel). Remember that the normalizer is multiplied by the sum of the convolution and not divided into it.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. (See above.) However, before ImageGear performs the operation specified by this function, it will check to see if an internal NRA flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask.

1.3.1.2.18.28 IG_IP_crop

This function crops the image to the specified rectangle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_crop(
   HIGEAR hIGear,
   LPAT_RECT lpCropRect
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpCropRect	LPAT_RECT	Pointer to an AT_RECT struct specifying the rectangular portion of the image to keep. The remainder of the image is removed and discarded.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;     // Count of errs on stack upon ret from func
AT_DIMENSION nWidth, nHeight;  // Dimensions of the image
AT_RECT rcRect;          // Crop rectangle

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get dimensions of the image and initialize the crop rectangle
    IG_image_dimensions_get(hIGear, &nWidth, &nHeight, NULL);
    rcRect.left = 0;
    rcRect.top = 0;
    rcRect.right = nWidth / 2;
    rcRect.bottom = nHeight / 2;

    nErrcount = IG_IP_crop(hIGear, &rcRect);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

Only pixels that fall on or inside the lpCropRect rectangle will be kept in the resulting image. The dimensions of the resulting image are then same as that of the lpCropRect. The removed parts of the image are discarded.

If right or bottom bound of the rectangle falls beyond the image dimensions, the rectangle is clipped to the image bounds. The resulting image dimensions cannot be larger than the dimensions of the source image. Use IG_IP_resize_canvas to extend the image bounds without scaling the image.

1.3.1.2.18.29 IG_IP_decrypt

This function decodes an image, or a rectangular portion thereof, that was encrypted using IG_IP_encrypt().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_decrypt (
        HIGEAR hIGear,
        LPAT_RECT lpRect,
        AT_MODE nEncryptType,
        LPSTR lpszPassword
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to be decoded.
lpRect	LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to decode. Set = NULL for the whole image.
nEncrptyType	AT_MODE	An IG_ENCRYPT_METHOD_ constant specifying how the image was encoded.
lpszPassword	LPSTR	Far pointer to your zero-terminated password string.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;
/* Decrypt the whole hIGear image by method A & with "Top Secret" password   */
IG_IP_decrypt (hIGear, NULL, IG_ENCRYPT_METHOD_A, "Top Secret");
```

Remarks:

This function works on a DIB, not on a file. You must supply both the encryption method and the password that were used in the call to IG_IP_encrypt().

1.3.1.2.18.30 IG_IP_deskew_angle_find

This function determines the skew angle of a 1-bit document image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_deskew_angle_find (
        HIGEAR hIGear,
        LPDOUBLE lpAngle
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpAngle	LPDOUBLE	Far pointer to a DOUBLE variable that will receive the skew angle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB – 1 bpp;
Grayscale – 1 bpp.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of image    */
DOUBLE dblDeskewAngle;  /* Amount of skew in lpRect returned  */
AT_ERRCOUNT  nErrcount;  /* will tally any IG errors  */
nErrcount = IG_IP_deskew_angle_find ( hIGear, &dblDeskewAngle );
```

Remarks:

The angle is returned via the pointer lpAngle. You may then use the value of lpAngle with IG_IP_rotate_any_angle() function in order to straighten the image.

1.3.1.2.18.31 IG_IP_deskew_auto

This function automatically detects the angle of the 1-bit document referenced by hIGear, and rotates it so that it is straight (i.e., de-skews it).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_deskew_auto (
        HIGEAR hIGear,
        DOUBLE dblAngleThresh,
        AT_MODE nExpand_clip_option
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to be decoded.
dblAngleThresh	DOUBLE	Do not de-skew if the skew angle is less than this parameter.
nExpand_clip_option	AT_MODE	An AT_MODE type variable, either: IG_ROTATE_CLIP or IG_ROTATE_EXPAND.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB – 1 bpp;
Grayscale – 1 bpp.

Example:

```
HIGEAR    hIGear;      /* HIGEAR handle of image  */
DOUBLE   angle_thresh;/* max angle above which should not perform skew  */
AT_MODE nExpand_clip_option;/* expand image or clip doc  */
AT_ERRCOUNT nErrcount;    /* will tally returned IG errors */
nErrcount = IG_IP_deskew_auto ( hIGear, 3.0, IG_ROTATE_CLIP);
```

Remarks:

If the skew angle is less than dblAngleThresh, then the image is not de-skewed. If nExpand_clip_option is set to IG_ROTATE_EXPAND, the size of image width or height will be enlarged as necessary to accommodate the document when it has been rotated. Otherwise, any areas of the document that now fall outside the borders of the original width and height of the image, will be cropped.

If you want to first detect the angle and then decide whether to rotate it, you can make separate calls to IG_IP_deskew_angle_find() and IG_IP_rotate_any_angle() functions.

1.3.1.2.18.32 IG_IP_despeckle

Despeckle is used to help reduce the amount of noise in the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_despeckle (
        HIGEAR hIGear,
        const LPAT_RECT lpRect
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate on. If NULL, the operation will be performed on the entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed images with non-grayscale palette.

Example:

```
HIGEAR hIGear;  /* HIGEAR handle of image */
nErrCount        = IG_IP_despeckle ( hIGear, NULL );
```

Remarks:

Single pixels and pixel spurs on letters and graphics are removed while leaving the solid areas alone. It is typically used on 1-bit document images.

The despeckle operation performs a 3x3 median filter on the image. For each 3x3 neighborhood of pixels in the original image, a single pixel is produced in the output image. In this case the output is the median of the 9 values in the 3x3 neighborhood.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.33 IG_IP_draw_frame

This function adds a frame (block of solid color on all four sides) to the image referenced by hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_draw_frame (
        HIGEAR hIGear,
        AT_DIMENSION width,
        AT_MODE nMethod,
        LPAT_PIXEL lpColor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
width	AT_DIMENSION	A variable that holds the width of the frame, in pixels.
nMethod	AT_MODE	A variable of type AT_MODE that tells ImageGear which IG_DRAW_FRAME_ setting to use. See Remarks.
lpColor	LPAT_PIXEL	A far pointer to the RGB value that specifies the frame's color.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_ERRCOUNT nErrcount;  /* holds tally of IG errors */
AT_DIMENSION width;  /* width, in pixels, of frame  */
AT_MODE           nMethod;  /* expand or overwrite image  */
AT_PIXEL lpColor[256];  /* color of frame; RGB value  */
nErrcount = IG_IP_draw_frame ( hIGear, 5, IG_DRAW_FRAME_EXPAND, &lpColor[9]);
```

Remarks:

If nMethod is set to IG_DRAW_FRAME_EXPAND, the width and the height of the image will be expanded by 2 times the width of the frame. If nMethod is set to IG_DRAW_FRAME_OVERWRITE, all four sides of the image will be overwritten by the frame, so that the resulting image has the same width and height as the original image, but the edges of the image are "covered" by the frame.

1.3.1.2.18.34 IG_IP_drop_shadow

This function adds a drop shadow to an image, which is enlarged to contain a background and shadowed area.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_drop_shadow(
        HIGEAR hIGear,
        AT_INT width,
        AT_INT distance,
        AT_INT angle,
        HIGPIXEL hInsideColor,
        HIGPIXEL hOutsideColor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image to which to apply drop shadow effect.
width	AT_INT	Width of background and shadow area to add to the image.
distance	AT_INT	Distance to move image from shadow area.
angle	AT_INT	Angle (direction) of movement of image from shadow area.
hInsideColor	HIGPIXEL	Color of shadow area. It should have the same color space and channel depths as the image to which the effect is applied.
hOutsideColor	HIGPIXEL	Color of background. It should have the same color space and channel depths as the image to which the effect is applied.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB – 1 bpp;
Indexed RGB with non-grayscale palette;
Grayscale – 1 bpp;
Images that have a Grayscale LUT attached to them.

1.3.1.2.18.35 IG_IP_edge_detection

This function performs the edge detection operation specified by the edge_detection_type argument.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_edge_detection(
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const AT_MODE edge_detection_type
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Handle of the input image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image on which to operate. If NULL, this operation will be performed on the entire image. Before ImageGear performs this operation it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
edge_detection_type	const AT_MODE	Type of edge detection method to perform. Constants IG_EDGE_DETECTION_* are listed in file accucnst.h.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Grayscale – 8 bpp.

Example:

```
HIGEAR hIGear;    /* Handle of the image */
AT_RECT lpRect;          /* Rectangle to process */
...
IG_IP_edge_detection(hIGear, lpRect, IG_EDGE_DETECTION_MAXGRADIENT);
...
```

Remarks:

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.36 IG_IP_edge_map

This function performs the image processing operation specified by the nEdgeMapTypeargument.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_edge_map (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const AT_MODE nEdgeMapType
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of an image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image on which to operate. If NULL, this operation will be performed on the entire image. Before ImageGear performs this operation it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nEdgeMapType	const AT_MODE	Type of edge map operation to perform, such as IG_EDGE_OP_LAPLACIAN, IG_EDGE_OP_ROBERTS, etc. The list of available types is in the accucnst.h.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR   hIGear;   /* HIGEAR handle of an image *?
IG_IP_edge_map ( hIGear, NULL, IG_EDGE_OF_ALPLACIAN ):
```

Remarks:

The operation is performed upon the rectangular portion of the image specified by lpRect.

An edge map is an image that shows where there are changes in contrast in the original image. Where there are no changes, the resulting image is black. The stronger the contract change, the brighter the resultant image. The different types of edge maps are slight variations of the algorithm used, and produce slightly different results.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image.If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask ().

See the descriptions of IG_IP_NR_ROI_mask_associate() and IP_IP_NR_ROI_to_HIGEAR_mask () functions for more details.

1.3.1.2.18.37 IG_IP_encrypt

This function scrambles an image bitmap, or a rectangular portion thereof.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_encrypt (
        HIGEAR hIGear,
        LPAT_RECT lpRect,
        AT_MODE nEncryptType,
        const LPSTR lpszPassword
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to be encoded.
lpRect	LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to encode. Set = NULL for the whole image.
nEncrptyType	AT_MODE	An IG_ENCRYPT_METHOD_ constant specifying the method to be used. See file accucnst.h for IG_ENCRYPT_METHOD_ constants available.
lpszPassword	const LPSTR	Far pointer to your zero-terminated password string.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;
/* Encrypt the whole hIGear image by method A & with "Top Secret" password   */
IG_IP_encrypt ( hIGear, NULL, IG_ENCRYPT_METHOD_A, "Top Secret" );
```

Remarks:

Your password is also stored. To later decode the image using IG_IP_decrypt(), you will need to know both the encryption method and the password used in this call.

This function cannot take a non-rectangular ROI for its AT_RECT parameter.

1.3.1.2.18.38 IG_IP_enhance_local

This function enhances an image using the local standard deviation and mean.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_enhance_local(
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const AT_DIMENSION nWinWidth,
        const AT_DIMENSION nWinHeight,
        const AT_DOUBLE dScaleFactor,
        const AT_DOUBLE dMinStdDev
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to be processed.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image on which to operate. If NULL, this operation will be performed on the entire image. Before ImageGear performs this operation it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nWinWidth	const AT_DIMENSION	Width of the local window.
nWinHeight	const AT_DIMENSION	Height of the local window.
dScaleFactor	const AT_DOUBLE	Scaling factor.
dMinStdDev	const AT_DOUBLE	Minimum allowed standard deviation.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Grayscale – 8 bpp.

Example:

```
HIGEAR hIGear;   /* HIGEAR handle of image */
AT_RECT lpRect;          /* rectangle to process */
AT_DIMENSION nWinWidth; /* Window width */
AT_DIMENSION    nWinHeight;     /* Window height */
AT_DOUBLE dScaleFactor; /* Tuning factor */
AT_DOUBLE dMinStdDev; /* Minimum allowed standard deviation */
...
IG_IP_enhance_local(hIGear, lpRect, nWinWidth, nWinHeight, dScaleFactor, dMinStdDev);
...
```

Remarks:

This function transforms the input image f(x, y) to a new image g(x, y) based on the following formula,

```
g(x, y) = A(x, y) * [f(x, y) - m(x, y)] + m(x, y)
```

where,

A(x, y) = k * M / sigma(x, y), with k being a scaling factor within the range [0, 1], m(x, y) and sigma(x, y) being the local mean and local standard deviation, and M being the global mean of the input image.

To avoid the problem of spikes caused by too small local standard deviation, a check against the minimum allowed standard deviation is performed. If the local standard deviation is too small, the minimum allowed will instead be used in the calculation.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.39 IG_IP_find_tilt

This function computes the least-squares best fit plane for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_find_tilt(
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        LPAT_DOUBLE lpSlopeX,
        LPAT_DOUBLE lpSlopeY,
        LPAT_DOUBLE lpPiston
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to be processed.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image on which to operate. If NULL, this operation will be performed on the entire image. Before ImageGear performs this operation it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
lpSlopeX	LPAT_DOUBLE	Returns the slope in the X direction of the tilt plane.
lpSlopeY	LPAT_DOUBLE	Returns the slope in the Y direction of the tilt plane.
lpPiston	LPAT_DOUBLE	Returns the piston of the tilt plane.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Grayscale - 8, 16, 32 bpp.

Example:

```
HIGEAR   hIGear;          /* HIGEAR handle of image */
AT_RECT lpRect;          /* rectangle to process */
LPAT_DOUBLE lpSlopeX, lpSlopeY, lpPiston;
...
IG_IP_find_tilt(hIGear, lpRect, lpSlopeX, lpSlopeY, lpPiston);
...
```

Remarks:

The plane will be given by formula f(x, y) = SlopeX * x + SlopeY * y + Piston.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.40 IG_IP_flip

Flips the image referenced by hIGear either horizontally or vertically.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_flip(
   HIGEAR hIGear,
   AT_MODE nDirection
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to flip.
nDirection	AT_MODE	IG_FLIP_HORIZONTAL or IG_FLIP_VERTICAL, indicating whether to flip horizontally or vertically.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_flip(hIGear, IG_FLIP_VERTICAL);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

Flipping horizontally exchanges the right-most pixel column of the image bitmap with the left-most. Flipping vertically exchanges the topmost pixel row (raster) of the image bitmap with the bottom-most. The dimensions of the image do not change.

If you want to turn the image upside-down (not the same as a vertical flip), use function IG_IP_rotate_multiple_90, with rotation mode set to IG_ROTATE_180.

1.3.1.2.18.41 IG_IP_gaussian_blur

This function smoothes the images using Gaussian transform.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_gaussian_blur (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const double dblRadius
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to AT_RECT struct specifying a portion of the image to be affected. NULL means entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
dblRadius	const double	Defines the neighborhood to be considered for each pixel.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette.

Example:

```
HIGEAR hIGear;
/* Blur whole image, radius = 2.0 */
nErrCount =IG_IP_gaussian_blur ( hIGear, NULL, 2.0 );
```

Remarks:

This function makes images look softer and slightly out of focus. A specific feature of Gaussian Blur is that it removes the high-frequency component from the image, which is not the case for the IG_IP_smooth().

Parameter dblRadius corresponds to the Standard Deviation (Sigma) in Gaussian transform. It can range from 0.1 to 500. Typical values for high-resolution images range from 1.0 to 2.0. Larger values will cause greater softening. On the other hand, smaller values are faster. The width of the area considered for each pixel is approximately 6 * dblRadius.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

1.3.1.2.18.42 IG_IP_geom_despeckle

This function is used to reduce speckle noise from an image by using the Crimmins algorithm.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_geom_despeckle(
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const AT_INT nIterations
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to be processed.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image on which to operate. If NULL, this operation will be performed on the entire image. Before ImageGear performs this operation it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nIterations	const AT_INT	Number of iterations to apply the despeckle filter.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Grayscale – 8 bpp.

Example:

```
HIGEAR   hIGear; /* HIGEAR handle of image */
AT_RECT lpRect; /* rectangle to process */
AT_INT nIterations;     /* Number of iterations */
...
IG_IP_geom_despeckle(hIGear, lpRect, nIterations);
...
```

Remarks:

This function reduces the speckle index of an image by sending the image through a geometric filter, which uses the complementary hulling technique. The method has the effect of reducing the undesired speckle noise while preserving the edges of the original image.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.43 IG_IP_histo_clear

This function will clear an array of histogram bins.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_histo_clear (
        LPDWORD lpHisto,
        UINT nNumberOfBins
);
```

Arguments:

Name	Type	Description
lpHisto	LPDWORD	Far pointer to an array of bins (each a DWORD) to be cleared.
nNumberOfBins	UINT	Number of bins to clear.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

See the example under the IG_IP_histo_tabulate() function.

Remarks:

Call this function prior to calling IG_IP_histo_tabulate(), unless you mean to accumulate onto existing contents of the bins.

1.3.1.2.18.44 IG_IP_histo_tabulate

This function produces a histogram of the pixel values occurring in image hIGear, or in the rectangular portion specified if lpRect is not NULL.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_histo_tabulate (
        HIGEAR hIGear,
        LPDWORD lpHisto,
        UINT nNumberOfBins,
        LPAT_RECT lpRect,
        UINT nYIncr,
        const AT_MODE nColorChannel
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpHisto	LPDWORD	Far pointer to an array of bins (each a DWORD) in which to tabulate.
nNumberOfBins	UINT	Number of bins.
lpRect	LPAT_RECT	Far pointer to an AT_RECT struct specifying rectangular portion of the image to tabulate for, or NULL for whole image. an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nYIncr	UINT	Set = 1 to sample every pixel. Values larger than 1 will skip rasters of the image bitmap.
nColorChannel	const AT_MODE	For 24 bit images selects which channel: IG_COLOR_COMP_R, _G, or _B (IG_COLOR_COMP_RGB not allowed).

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;    /* HIGEAR handle of image */
DWORD dwHistoBins[256];   /* Array of bins for counting */
IG_IP_histo_clear ( &dwHistoBins, 256 ); /* Clear the bins */
/* Tabulate. If a 24-bit image, only the red will be tabulated: */
IG_IP_histo_tabulate (hIGear, &dwHistoBins, 256, NULL, 1, IG_COLOR_COMP_R);
```

Remarks:

If nYIncr = 1, each pixel in the image is examined. The bin corresponding to that pixel value increments. The number of bins must be large enough to hold the entire histogram. If an 8 or 24-bit image, the number of bins must be 256. If a 4-bit or 1-bit image, the number of bins must be 16 or 2 respectively. Note that for a 24-bit image, only a single color channel can be a histogram at one time. Use argument nColorChannel to select the channel.

If nYIncr is greater than 1, then rasters of the image are skipped. This can be used to speed up the tabulation when the image is large.

You should call IG_IP_histo_clear() before calling this function, unless you intentionally mean to accumulate the count onto the existing contents of the bins.

1.3.1.2.18.45 IG_IP_maximum

This function performs a maximum filter on an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_maximum (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        AT_DIMENSION nNeighborWidth,
        AT_DIMENSION nNeighborHeight
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate upon. If NULL, the entire image will be operated upon. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nNeighborWidth	AT_DIMENSION	Width of the neighborhood to include in computing each new pixel's value. Must be positive.
nNeighborHeight	AT_DIMENSION	Height of the neighborhood to include in computing each new pixel's value. Must be positive.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image */
nErrCount = IG_IP_maximum ( hIGear, NULL, 5, 5 );
```

Remarks:

A maximum filter makes the lighter pixels larger and shrinks the darker ones. The width and height determine the size of each original pixel's neighborhood to use to compute the maximum output pixels. Most applications will find that 3x3 or 5x5 works best.

Only the lpRect of the image is processed. Set lpRect = NULL to process the entire image.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.46 IG_IP_median

This function performs a median filter on an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_median (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        AT_DIMENSION nNeighborWidth,
        AT_DIMENSION nNeighborHeight
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT struct specifying the rectangular portion of the image to operate upon. If NULL, the entire image will be operated upon. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nNeighborWidth	AT_DIMENSION	Width of the neighborhood to include in computing each new pixel's value. Must be positive.
nNeighborHeight	AT_DIMENSION	Height of the neighborhood to include in computing each new pixel's value. Must be positive.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette.

Example:

```
HIGEAR hIGear;     /* HIGEAR handle of image */
nErrCount         = IG_IP_median ( hIGear, NULL, 5, 5 );
```

Remarks:

A median filter is useful for reducing spike or snow-like noise from an image. The width and height determine the size of each original pixel's neighborhood to use to compute the median output pixels. Most applications will find that 3x3 or 5x5 works best.

Only the lpRect of the image is processed. Set lpRect = NULL to process the entire image.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.47 IG_IP_merge

This function is used to "place" or merge one image into another.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_merge (
        HIGEAR hImage1,
        HIGEAR hImage2,
        LPAT_RECT lpImageRect2,
        AT_PIXPOS nDstX,
        AT_PIXPOS nDstY,
        AT_MODE nPix_op
);
```

Arguments:

Name	Type	Description
hImage1	HIGEAR	HIGEAR handle of image.
hImage2	HIGEAR	HIGEAR image to be merged into hImage1; must have same bit depth as hImage1.
lpImageRect2	LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of hImage2 to merge into hImage1. upon. Set to NULL if you want to merge the entire image of hImage2. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non - rectangular ROI defined by the mask.
nDstX	AT_PIXPOS	The x coordinate within hImage1 at which to place the upper - left corner of hImage2.
nDstY	AT_PIXPOS	The y coordinate within hImage1 at which to place the upper-left corner of hImage2.
nPix_op	AT_MODE	A variable of constant type AT_MODE that specifies what type of arithmetic operation (merge method) to perform on all pixels of hImage1 that have been intersected with pixels from hImage2. Examples are IG_ARITH_ADD, which adds the pixel values of both images, and IG_ARITH_SUB, which subtracts the pixel values of hImage2 from the corresponding pixel values of hImage1. For the full list of available constants, see accucnst.h or see the description for IG_clipboard_paste_op_set() which also uses these constants for full list.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear1,  /* HIGEAR handle of destination image   */
                hIGear2;  /* HIGEAR handle of image to be merged in */
AT_ERRCOUNT nErrcount; /* # of ImageGear errors on stack */
nErrcount = IG_IP_merge ( hIGear1, hIGear2, NULL, 0, 0, IG_ARITH_OVER );
```

Remarks:

The images do not have to be the same size but do have to be the same bit depth. hImage2 is drawn into hImage1. The top left corner of hImage2 will be placed at nDstX, nDstY of hImage1. Any over-hanging pixels of hImage2 will be clipped automatically. The nPix_op, which is defined in accucnst.h, determines how the pixels are combined.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can merge a rectangular sub-region of HIGEAR2 into HIGEAR1. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

if lpImageRect2 is not NULL, or hIGear2 has a mask attached to it, and you want to place the top left corner of lpImageRect2 or the mask at nDstX, nDstY of hImage1, subtract the coordinates of the left top corner of lpImageRect2 or mask from the destination coordinates.

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.48 IG_IP_minimum

This function performs a minimum filter on an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_minimum (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        AT_DIMENSION nNeighborWidth,
        AT_DIMENSION nNeighborHeight
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image to operate upon. If NULL, the entire image will be operated upon. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
nNeighborWidth	AT_DIMENSION	Width of the neighborhood to include in computing each new pixel's value. Must be positive.
nNeighborHeight	AT_DIMENSION	Height of the neighborhood to include in computing each new pixel's value. Must be positive.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette.

Example:

```
HIGEAR  hIGear;    /* HIGEAR handle of image */
nErrCount = IG_IP_minimum ( hIGear, NULL, 5, 5 );
```

Remarks:

A minimum filter makes the lighter pixels smaller and the darker ones larger. The width and height determine the size of each original pixel's neighborhood to use to compute the minimum output pixels. Most applications will find that 3x3 or 5x5 works best.

Only the lpRect of the image is processed. Set lpRect = NULL to process the entire image.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear overrides the settings passed to the AT_RECT structure and uses the non-rectangular ROI defined by the HIGEAR mask. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.49 IG_IP_NR_ROI_control_get

This function will return to you the current setting of any of the non-rectangular ROI control settings.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_control_get(
        HIGEAR hIGear,
        AT_MODE  nAttributeID,
        VOID FAR32* lpData
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image for which you would like to query ROI settings.
nAttributeID	AT_MODE	Set to an AT_MODE constant for the type of attribute you wish to query. See Remarks.
lpData	VOID FAR32*	This returns the current setting of nAttributeID.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT      nErrcount;
HIGEAR  hIGear;
BOOL    bUseNonRect;
AT_POINT        ptReferencePoint;
/* Find out whether the image rectangle for hIGear is set to be overridden by a non-
rectangular ROI    */
nErrcount = IG_IP_NR_ROI_control_get (hIGear, IG_CONTROL_NR_ROI_STATE, &bUseNonRect);
/* Find out what the reference point for a mask HIGEAR is in the hIGear */
nErrcount = IG_IP_NR_ROI_control_get (hIGear, IG_CONTROL_NR_ROI_REFERENCE_POINT,
&ptReferencePoint);
```

Remarks:

Supply ImageGear with the HIGEAR handle of the image you are querying and the attribute (nAttributeID) whose setting you would like to query. The ROI settings currently available are:

- IG_CONTROL_NR_ROI_DIB: Returns the DIB which is currently set to be used as the mask HIGEAR.
- IG_CONTROL_NR_ROI_REFERENCE_POINT: Returns the reference point of the mask.
- IG_CONTROL_NR_ROI_REFERENCE_POINT_LEFT: Returns left point of the mask.
- IG_CONTROL_NR_ROI_REFERENCE_POINT_TOP: Returns top point of the mask.
- IG_CONTROL_NR_ROI_CONDITION: Queries the "condition" of the mask HIGEAR: whether it is set to be active or not active for the next IP or Clipboard operation.
- IG_CONTROL_NR_ROI_VALIDATE: Returns TRUE if the current ROI mask is valid.

To change these settings and for details on how these controls can be used, see the description for IG_IP_NR_ROI_control_set().

1.3.1.2.18.50 IG_IP_NR_ROI_control_set

This function allows you to set the non-rectangular ROI attributes associated with any HIGEAR that has an associated non-rectangular ROI mask.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_control_set(
        HIGEAR hIGear,
        AT_MODE nAttributeID,
        const LPVOID lpData
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image to which you wish make the non-rectangular ROI setting changes.
nAttributeID	AT_MODE	Set to an AT_MODE constant for the non-rectangular ROI attribute you would like to set.
lpData	const LPVOID	Set to the desired attribute setting.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT      nErrcount;
HIGEAR  hIGear;
BOOL    bCondition;
HIGEAR  hIGearMask;]
/* Find out whether the image rectangle for hIGear is set to be overridden by a non-
rectangular ROI   */
nErrcount = IG_IP_NR_ROI_control_set (hIGear, IG_CONTROL_NR_ROI_DIB, (LPVOID) hIGearMask);
/* Find out what the reference point for a mask HIGEAR is in the hIGear */
nErrcount = IG_IP_NR_ROI_control_set (hIGear, IG_CONTROL_NR_ROI_CONDITION, (LPVOID)
bCondition);
```

Remarks:

These attributes are only applicable for non-rectangular ROIs. Use IG_IP_NR_ROI_to_HIGEAR_mask() to create the mask image.

All ROI control settings have defined constants in accucnst.h which have a prefix of IG_CONTROL_NR_ROI_. The following is a list of each setting available at the time of this writing, and a description of what each does.

- IG_CONTROL_NR_ROI_DIB: Sets the DIB to be used as the mask HIGEAR for the currently loaded HIGEAR image.
- IG_CONTROL_NR_ROI_REFERENCE_POINT: Sets the position within the HIGEAR image at which the upper-left corner of the masking HIGEAR should be placed.
- IG_CONTROL_NR_ROI_REFERENCE_POINT_LEFT: Sets the left point of the mask.
- IG_CONTROL_NR_ROI_REFERENCE_POINT_TOP: Sets the top point of the mask.
- IG_CONTROL_NR_ROI_CONDITION: Sets whether or not ImageGear should override the AT_RECT argument passed to its API. Set to TRUE if you would like ImageGear to use the non-rectangular ROI defined by the mask

HIGEAR. Set to FALSE for ImageGear to use the rectangular ROI defined by the current image rectangle.

IG_CONTROL_NR_ROI_CONDITION can also be set using IG_IP_NR_ROI_mask_associate() function.

1.3.1.2.18.51 IG_IP_NR_ROI_mask_associate

This function will associate a mask HIGEAR, as specified by the AT_NR_ROI_MASK structure, with the image referenced by hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_mask_associate(
        HIGEAR hIGear,
        LPAT_NR_ROI_MASK lpMask,
        BOOL bState
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image to associate with a non-rectangular ROI mask.
lpMask	LPAT_NR_ROI_MASK	Pass ImageGear a long pointer to a structure of type AT_NR_ROI_MASK that gives the HIGEAR handle of the mask HIGEAR.
bState	BOOL	Set to TRUE if you want to make the mask "active", meaning that affected IP or clipboard operations will operate on the non-rectangular region only; set to FALSE if you want affected IP or clipboard operations to apply the image rectangle specified by the AT_RECT argument.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT      nErrcount;
HIGEAR  hIGear;
AT_NR_ROI_MASK   lpMask;
BOOL    bState;
nErrcount = IG_IP_NR_ROI_mask_associate(hIGear, &lpMask, TRUE);
```

Remarks:

The bState argument determines whether or not the ROI defined by AT_NR_ROI_MASK should be made active. If you set bState to TRUE, ImageGear will override the AT_RECT argument passed to certain Image Processing and Clipboard API functions in favor of using the non-rectangular ROI. For example, the function IG_IP_contrast_adjust() takes an AT_RECT as an argument, so that you can adjust the contrast in a rectangular sub-region of an image, or adjust the contrast of the whole image (if you set the AT_RECT parameter to NULL). If you set bState to TRUE, when you next call IG_IP_contrast_adjust(), its AT_RECT argument will be ignored, or "overridden", and ImageGear will instead use the ROI described by the mask HIGEAR.

If you provide an invalid mask HIGEAR, you will receive the error IGE_INVALID_MASK_ASSOCIATED.

No change will take place in the image until you perform an Image Processing or Clipboard operation. When using image processing functions the changes made are permanent if you save the image. For this reason, you may want to keep a copy of the original image so that the user can "undo" an operation.

- The setting for bState, which ImageGear stores with the image can also be set using IG_IP_NR_ROI_control_set() with the constant IG_CONTROL_NR_ROI_CONDITION.
- To reset the reference point in HIGEAR, call IG_IP_NR_ROI_control_set() with the constant

IG_CONTROL_NR_ROI_REFERENCE_POINT.
- Call IG_IP_NR_ROI_mask_unassociate() to clear the mask HIGEAR from its association with HIGEAR.

To create a non-rectangular region of interest call IG_IP_NR_ROI_to_HIGEAR_mask() function.

1.3.1.2.18.52 IG_IP_NR_ROI_mask_delete

This function deletes the mask HIGEAR created by IG_IP_NR_ROI_to_HIGEAR_mask().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_mask_delete (
        LPAT_NR_ROI_MASK lpMask
);
```

Arguments:

Name	Type	Description
lpMask	LPAT_NR_ROI_MASK	A far pointer to the mask structure to delete.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Example:

```
AT_ERRCOUNT  nErrcount;
HIGEAR hIGear;
AT_NR_ROI_MASK Mask
nErrcount = IG_IP_NR_ROI_mask_delete(&Mask);
```

When you are done using this mask, and have called the function IG_IP_NR_ROI_mask_unassociate(), you should call IG_IP_NR_ROI_mask_delete() to delete the mask and free up the memory allocated to it.

See also IG_IP_NR_ROI_to_HIGEAR_mask(), IG_IP_NR_ROI_mask_associate(), and IG_IP_NR_ROI_mask_unassociate() functions.

1.3.1.2.18.53 IG_IP_NR_ROI_mask_unassociate

This function clears the non-rectangular ROI information from a HIGEAR image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_mask_unassociate(HIGEAR hIGear);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image for which you would like to remove the association to the mask HIGEAR.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT  nErrcount;
HIGEAR hIGear;
nErrcount = IG_IP_NR_ROI_mask_unassociate(hIGear);
```

Remarks:

This function does not delete the mask HIGEAR, it only removes the reference to it from this HIGEAR. To delete the mask HIGEAR, call IG_IP_NR_ROI_mask_delete().

See the description for IG_IP_NR_ROI_mask_associate() function for more information.

To delete the mask HIGEAR, use IG_image_delete() function.

1.3.1.2.18.54 IG_IP_NR_ROI_to_HIGEAR_mask

This function is a non-rectangular ROI (region on interest) support function whose purpose is to build a non-rectangular ROI mask from a set of segment descriptors that you pass in, and to return a pointer to a non-rectangular ROI mask data structure.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_NR_ROI_to_HIGEAR_mask(
        AT_MODE nSimpleAreaTypeID,
        LPVOID lpAreaSegmentDesc,
        LPAT_NR_ROI_MASK lpNR_ROI
);
```

Arguments:

Name	Type	Description
nSimpleAreaTypeID	AT_MODE	Set to an AT_MODE constant that describes what kind of non-rectangular region of interest (ROI) you will be passing in. See supported AT_MODE constants below.
lpAreaSegmentDesc	LPVOID	Pass in an array of segment descriptors which will be used to reproduce/render the non-rectangular ROI as a mask. These segment descriptors can be points in the case of polygons, points and angles in the case of ellipses and so on. The segment descriptors use image coordinates to describe the mask.
lpNR_ROI	LPAT_NR_ROI_MASK	ImageGear returns you a structure of type AT_NR_ROI_MASK which contains the HIGEAR handle of the new mask HIGEAR and the "reference point" for its placement within HIGEAR.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT       nErrcount;
HIGEAR  hIGear;
AT_NR_ROI_MASK   region;
AT_POINT        ROI[];
nErrcount =
        IG_IP_NR_ROI_to_HIGEAR_mask(IG_ROI_IS_POLYGON, &ROI[0], &region);
```

Remarks:

This non-rectangular ROI mask data structure (of type AT_NR_ROI_MASK) is then used in conjunction with other API functions to create, associate, modify, and apply the non-rectangular ROI to the source image.

The AT_NR_ROI_MASK structure, shown below, contains two important pieces of information. ptMaskOffset describes the coordinates for reference point for the mask HIGEAR. ptMaskOffset is the (x,y) position in the original HIGEAR at which the upper left corner of the mask HIGEAR should be placed. The mask HIGEAR is actually a rectangle which is calculated by determining the smallest rectangular area that can encompass the entire non-rectangular ROI. We refer to this area as the "bounding rectangle". Within the mask, which represents the bounding rectangle, a pixel value of 1 indicates that the pixel is within the non-rectangular ROI; a pixel value of 0 indicates that the pixel is outside the non-rectangular ROI.

```
typedef struct tag     AT_NR_ROI_MASK
{
        AT_POINT ptMaskOffet;
        HIGEAR hMask;
}   AT_NR_ROI_MASK, FAR* LPAT_NR_ROI_MASK;
```

The second member of the mask structure is a HIGEAR handle to the actual mask image. The mask HIGEAR is a run length-encoded binary image.

You must also pass this function an argument that specifies whether the region of interest is elliptical, polygonal, or rectangular (the default), using one of the following constants:

- IG_ROI_IS_RECTANGLE
- IG_ROI_IS_ELLIPSE
- IG_ROI_IS_POLYGON

The default ROI type, which is IG_ROI_IS_RECTANGLE means that when this mask is associated with and activated for an image, all affected API should use the AT_RECT argument that is part of their argument list. If nSimpleAreaTypeID is set to IG_ROI_IS_ELLIPSE or IG_ROI_IS_POLYGON, all affected API will override their AT_RECT arguments and instead look for an associated AT_NR_ROI_MASK.

The lpNR_ROI parameter will store the mask HIGEAR information structure when the function return value is IGE_SUCCESS.

Use IG_IP_NR_ROI_mask_associate() function to associate the mask HIGEAR with a HIGEAR image.

1.3.1.2.18.55 IG_IP_pseudocolor_limits

This function colors all pixels in an 8-bit gray level image whose values are outside the range nLow to nHigh.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_pseudocolor_limits (
        HIGEAR hIGear,
        LPAT_RGB lpRGB_Low,
        LPAT_RGB lpRGB_High,
        AT_PIXEL nLow,
        AT_PIXEL nHigh
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of 8-bit grayscale image to be colored.
lpRGB_Low	LPAT_RGB	Far pointer to an AT_RGB struct (note: B-G-R) specifying the color to apply to all pixels below the nLow value.
lpRGB_High	LPAT_RGB	Far pointer to an AT_RGB struct (note: B-G-R) specifying the color to apply to all pixels above the nHigh value.
nLow	AT_PIXEL	All pixels below this value are colored.
nHigh	AT_PIXEL	All pixels above this value are colored.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Grayscale – 8-16 bpp.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image  */
AT_RGB cLowColor, cHighColor; /* The colors to apply */
AT_PIXEL nLow, nHigh;  /* Where to apply them  */
cLowColor.b  = cLowColor.g  = 0;  cLowColor.r  = 255; /* bright red  */
cHighColor.b = cHighColor.r = 0;  cHighColor.g = 255; /* bright green*/
        /* Retain image colors for pixel values 25 through 225:     */
nLow = 25;  nHigh = 225;
IG_IP_pseudocolor_limits ( hIGear, &cLowColor, &cHighColor, nLow, nHigh );
```

Remarks:

Those values above the range receive the color pointed to by lpRGB_High, and those below the range receive the color pointed to by lpRGB_Low. This function can be used to see how much of the image is saturated or unsaturated.

Remember that the order in the RGB structure is B-G-R. See the section Device-Independent Bitmaps (DIB) for more information.

1.3.1.2.18.56 IG_IP_pseudocolor_small_grads

This function colors an 8-bit gray level image such that small gradients are exposed.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_pseudocolor_small_grads (
        HIGEAR hIGear,
        UINT nSlope
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of 8-bit grayscale image to be colored.
nSlope	UINT	An integer from 1 to 255. Higher values increase colors faster.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Grayscale – 8-16 bpp.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Example:

```
HIGEAR hIGear;
if ( IG_image_is_valid(hIGear) )
        if ( IG_image_is_gray(hIGear) )
        IG_IP_pseudocolor_small_grads ( hIGear, 10 );
```

Remarks:

The greater the value of nSlope, the faster the color will change for a given rate of change of the pixel value.

1.3.1.2.18.57 IG_IP_remove_tilt

This function computes the best-fit plane for an image, and then subtracts that plane from the image to produce the output.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_remove_tilt(
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        AT_BOOL bRemoveMean
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to be processed.
lpRect	const LPAT_RECT	Far pointer to an AT_RECT structure specifying the rectangular portion of the image on which to operate. If NULL, this operation will be performed on the entire image. Before ImageGear performs this operation it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
bRemoveMean	AT_BOOL	Remove the mean from the de-tilted image, giving it zero-mean statistics.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Grayscale - 8, 16, 32 bpp.

Example:

```
HIGEAR hIGear; /* HIGEAR handle of image */
AT_BOOL         bRemoveMean;    /* TRUE = remove mean */
AT_RECT         lpRect; /* rectangle to process */
...
IG_IP_remove_tilt( hIGear, lpRect, bRemoveMean);
...
```

Remarks:

This function is very handy for correcting illumination gradients in a poorly digitized image.

If bRemoveMean argument is set to TRUE, then remove mean from image.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.58 IG_IP_resize

This function resizes the image referenced by hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_resize(
   HIGEAR hIGear,
   AT_DIMENSION nNewWidth,
   AT_DIMENSION nNewHeight,
   AT_MODE nInterpMethod
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to resize.
nNewWidth	AT_DIMENSION	Width that the image is to be after resizing.
nNewHeight	AT_DIMENSION	Height that the image is to be after resizing.
nInterpMethod	AT_MODE	Specifies interpolation method to use for image resizing. See enumIGInterpolations for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If nInterpMethod is IG_INTERPOLATION_GRAYSCALE, IG_INTERPOLATION_PRESERVE_WHITE, or IG_INTERPOLATION_PRESERVE_BLACK:

- Indexed RGB - 1 bpp;
- Grayscale - 1 bpp.

If nInterpMethod is IG_INTERPOLATION_AVERAGE or IG_INTERPOLATION_BILINEAR:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.

If nInterpMethod is IG_INTERPOLATION_BICUBIC:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.
- Grayscale - 1 bpp.

Otherwise, all pixel formats supported by ImageGear Professional.

This function does not support PDF images.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func
AT_DIMENSION nWidth, nHeight;   // Dimensions of the image

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get dimensions of the image
    IG_image_dimensions_get(hIGear, &nWidth, &nHeight, NULL);

    nErrcount = IG_IP_resize(hIGear, nWidth / 2, nHeight / 2, IG_INTERPOLATION_NONE);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The image data in the bitmap will be stretched, compressed, or padded as necessary to fit the new dimensions.

During resizing, new pixel values that previously did not exist in the image may be introduced due to interpolation. If you want to prevent this, such as to preserve the original number of palette entries used, then specify IG_INTERPOLATION_NONE. In this case, only pixel values that occur in the original image will result in the resized image.

The functionality of this API call has been upgraded and supported by the new function IG_IP_resize_bkgrnd_ex. This new function allows you to change the background color around the image being resized, if the interpolation is either IG_INTERPOLATION_PADDING or IG_INTERPOLATION_CANVAS. In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

1.3.1.2.18.59 IG_IP_resize_bkgrnd

This function resizes the image referenced by hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_resize_bkgrnd(
   HIGEAR hIGear,
   AT_DIMENSION nNewWidth,
   AT_DIMENSION nNewHeight,
   AT_MODE nInterpMethod,
   LPAT_PIXEL lpBkgColor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to resize.
nNewWidth	AT_DIMENSION	Width that the image is to be after resizing.
nNewHeight	AT_DIMENSION	Height that the image is to be after resizing.
nInterpMethod	AT_MODE	Specifies interpolation method to use for image resizing. See enumIGInterpolations for possible values.
lpBkgColor	LPAT_PIXEL	Pointer to the RGB or pixel value that specifies the background color to be used in the displaced areas after the image has been resized when using the resize with padding or canvas method.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If nInterpMethod is IG_INTERPOLATION_GRAYSCALE, IG_INTERPOLATION_PRESERVE_WHITE, or IG_INTERPOLATION_PRESERVE_BLACK:

- Indexed RGB - 1 bpp;
- Grayscale - 1 bpp.

If nInterpMethod is IG_INTERPOLATION_AVERAGE or IG_INTERPOLATION_BILINEAR:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.

If nInterpMethod is IG_INTERPOLATION_BICUBIC:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.
- Grayscale - 1 bpp.

Otherwise, all pixel formats supported by ImageGear Professional.

This function does not support PDF images.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func
AT_DIMENSION nWidth, nHeight;  // Dimensions of the image
AT_INT channelCount;      // Count of channels in the image
AT_PIXEL lpBackground[256];    // Buffer for background color
```

```
AT_INT i;

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get dimensions of the image
    IG_image_dimensions_get(hIGear, &nWidth, &nHeight, NULL);
    // Get channel count
    IG_image_channel_count_get(hIGear, &channelCount);
    // Initialize background color with '255'
    for(i = 0; i < channelCount; i ++)
    {
        lpBackground[i] = (AT_PIXEL)255;
    }

    nErrcount = IG_IP_resize_bkgrnd(hIGear, nWidth / 2, nHeight / 2,
IG_INTERPOLATION_BILINEAR, lpBackground);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The image data in the bitmap will be stretched, compressed, or padded as necessary to fit the new dimensions.

During resizing, new pixel values that previously did not exist in the image may be introduced due to interpolation. If you want to prevent this, such as to preserve the original number of palette entries used, then specify IG_INTERPOLATION_NONE. In this case only pixel values that occur in the original image will result in the resized image.

Setting the IG_INTERPOLATION_PADDING means that if you increase the size of the image, it is padded to the new boundaries. Pixels added to the right and bottom of the original image will be filled with lpBkgColor. If you decrease the size of the image with IG_INTERPOLATION_PADDING, the image is cropped.

1.3.1.2.18.60 IG_IP_resize_bkgrnd_ex

This function resizes the image referenced by hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_resize_bkgrnd_ex(
   HIGEAR hIGear,
   AT_DIMENSION nNewWidth,
   AT_DIMENSION nNewHeight,
   AT_MODE nInterpMethod,
   LPAT_PIXEL lpBkgColor,
   DWORD dwFlags,
   INT nValue
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to resize.
nNewWidth	AT_DIMENSION	The width of the resized image.
nNewHeight	AT_DIMENSION	The height of the resized image.
nInterpMethod	AT_MODE	Specifies interpolation method to use for image resizing. See enumIGInterpolations for possible values.
lpBkgColor	LPAT_PIXEL	Pointer to the RGB or pixel value that specifies the background color to be used in the displaced areas after the image is resized using padding method.
dwFlags	DWORD	Reserved for future use.
nValue	INT	The contents of this parameter depends upon the value of nInterpMethod: • IG_INTERPOLATION_GRAYSCALE - nValue can be from 0 to 100. It takes the proportion of pixels from entry 1 to entry 0 (white/black). • IG_INTERPOLATION_PRESERVE_WHITE - nValue can be from 0 to 100. It indicates the threshold value of the amount of white color to include. • IG_INTERPOLATION_PRESERVE_BLACK - nValue can be from 0 to 100. It indicates the threshold value of the amount of black color to include. • Any other values - nValue is ignored.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If nInterpMethod is IG_INTERPOLATION_GRAYSCALE, IG_INTERPOLATION_PRESERVE_WHITE, or IG_INTERPOLATION_PRESERVE_BLACK:

- Indexed RGB - 1 bpp;
- Grayscale - 1 bpp.

If nInterpMethod is IG_INTERPOLATION_AVERAGE or IG_INTERPOLATION_BILINEAR:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.

If nInterpMethod is IG_INTERPOLATION_BICUBIC:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.
- Grayscale - 1 bpp.

Otherwise, all pixel formats supported by ImageGear Professional.

This function does not support PDF images.

Example:

```
HIGEAR hIGear;             // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;     // Count of errs on stack upon ret from func
AT_DIMENSION nWidth, nHeight;  // Dimensions of the image
AT_INT channelCount;     // Count of channels in the image
AT_PIXEL lpBackground[256];    // Buffer for background color
AT_INT i;

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get dimensions of the image
    IG_image_dimensions_get(hIGear, &nWidth, &nHeight, NULL);
    // Get channel count
    IG_image_channel_count_get(hIGear, &channelCount);
    // Initialize background color with '255'
    for(i = 0; i < channelCount; i ++)
    {
        lpBackground[i] = (AT_PIXEL)255;
    }

    nErrcount = IG_IP_resize_bkgrnd_ex(hIGear, nWidth / 2, nHeight / 2,
IG_INTERPOLATION_BILINEAR, lpBackground,
        0, 0);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The image data in the bitmap will be stretched, compressed, or padded as necessary to fit the new dimensions.

During resizing, new pixel values that previously did not exist in the image may be introduced due to interpolation. If you want to prevent this, (to preserve the original number of palette entries used, for example) specify IG_INTERPOLATION_NONE. In this case only pixel values that occur in the original image will result in the resized image.

Setting the IG_INTERPOLATION_PADDING means that if you increase the size of the image, it is padded to the new boundaries. Pixels added to the right and bottom of the original image will be filled with lpBkgColor. If you decrease the size of the image with IG_INTERPOLATION_PADDING, the image is cropped.

1.3.1.2.18.61 IG_IP_resize_canvas

This function resizes the image referenced by hIGear without scaling it.

Declaration:

```
AT_ERRCOUNT LACCUAPI IG_IP_resize_canvas(
   HIGEAR hIGear,
   AT_DIMENSION new_width,
   AT_DIMENSION new_height,
   AT_PIXPOS nXPos,
   AT_PIXPOS nYPos,
   LPAT_PIXEL lpBkgColor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image to process.
new_width	AT_DIMENSION	Width of the image after resizing.
new_height	AT_DIMENSION	Height of the image after resizing.
nXPos	AT_PIXPOS	X offset at which to put left top corner of the image after resizing.
nYPos	AT_PIXPOS	Y offset at which to put left top corner of the image after resizing.
lpBkgColor	LPAT_PIXEL	Color to fill the empty area. Ignored for vector images.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

The image data in the bitmap is not stretched or compressed, but copied to the specified offset in the new image.

1.3.1.2.18.62 IG_IP_resize_ex

This function resizes the image referenced by hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_resize_ex(
   HIGEAR hIGear,
   AT_DIMENSION nNewWidth,
   AT_DIMENSION nNewHeight,
   AT_MODE nInterpMethod,
   DWORD dwFlags,
   INT nValue
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to resize.
nNewWidth	AT_DIMENSION	Width to which the image is to be resized.
nNewHeight	AT_DIMENSION	Height to which the image is to be resized.
nInterpMethod	AT_MODE	Specifies interpolation method to use for image resizing. See enumIGInterpolations for possible values.
dwFlags	DWORD	Reserved for future use.
nValue	INT	The contents of this parameter depends upon the value of nInterpMethod: • IG_INTERPOLATION_GRAYSCALE - nValue can be from 0 to 100. It takes the proportion of pixels from entry 1 to entry 0 (white/black). • IG_INTERPOLATION_PRESERVE_WHITE - nValue can be from 0 to 100. It indicates the threshold value of the amount of white color to include. • IG_INTERPOLATION_PRESERVE_BLACK - nValue can be from 0 to 100. It indicates the threshold value of the amount of black color to include. • Any other values - nValue is ignored.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If nInterpMethod is IG_INTERPOLATION_GRAYSCALE, IG_INTERPOLATION_PRESERVE_WHITE, or IG_INTERPOLATION_PRESERVE_BLACK:

- Indexed RGB - 1 bpp;
- Grayscale - 1 bpp;

If nInterpMethod is IG_INTERPOLATION_AVERAGE or IG_INTERPOLATION_BILINEAR:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.

If nInterpMethod is IG_INTERPOLATION_BICUBIC:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.
- Grayscale - 1 bpp.

Otherwise, all pixel formats supported by ImageGear Professional.

This function does not support PDF images.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func
AT_DIMENSION nWidth, nHeight;  // Dimensions of the image

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get dimensions of the image
    IG_image_dimensions_get(hIGear, &nWidth, &nHeight, NULL);

    nErrcount = IG_IP_resize_ex(hIGear, nWidth / 2, nHeight / 2,
IG_INTERPOLATION_BILINEAR, 0, 0);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The image data in the bitmap will be stretched, compressed, or padded as necessary to fit the new dimensions.

During resizing, new pixel values that previously did not exist in the image may be introduced due to interpolation. If you want to prevent this, such as to preserve the original number of palette entries used, then specify IG_INTERPOLATION_NONE. In this case, only pixel values that occur in the original image will result in the resized image.

The functionality of this API call has been upgraded and supported by the new function IG_IP_resize_bkgrnd_ex. The reason that this new function has been created is that the old function does not allow you to change the background color around the image being resized, if the interpolation is either IG_INTERPOLATION_PADDING or IG_INTERPOLATION_CANVAS. In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

1.3.1.2.18.63 IG_IP_RGB_to_hue

This function will convert a 24-bit RGB value to a hue value.

Declaration:

```
AT_ERRCOUNT ACCUAPIIG_IP_RGB_to_hue (
        const LPAT_RGB lpRGB,
        LPDOUBLE lpHue
);
```

Arguments:

Name	Type	Description
lpRGB	const LPAT_RGB	A long pointer to a structure of type AT_RGB containing three bytes of color information.
lpHue	LPDOUBLE	A long pointer to a double containing the hue value that corresponds to the color specified by lpRGB.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

RGB – 24 bpp.

Example:

See the example for IG_FX_chroma_key() function.

Remarks:

You pass it a pointer to the AT_RGB structure of your choice, and it returns the hue angle (0.0 to 360) to you. This is useful before calling IG_FX_chroma_key() which requires a hue angle as one of its arguments. (ImageGear's pixel access functions only read and write RGB values.) You can use them to read the RGB value of a "background" pixel in the image which you wish to make transparent.

1.3.1.2.18.64 IG_IP_rotate_any_angle

This function rotates the image by the specified angle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_any_angle(
   HIGEAR hIGear,
   DOUBLE angle,
   AT_MODE rotate_mode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
angle	DOUBLE	Angle by which to rotate the image, in degrees. Positive values result in clockwise rotation; negative values result in counter-clockwise rotation.
rotate_mode	AT_MODE	Rotation mode. Specifies whether the image should be clipped or expanded. See enumIGRotationModes for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_rotate_any_angle(hIGear, 45., IG_ROTATE_CLIP);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The function rotates the image about its center point.

You can use IG_IP_rotate_compute_size to calculate the new dimensions of the bitmap that the image will have after rotation in IG_ROTATE_EXPAND mode.

The functionality of this API call has been upgraded and supported by the new function IG_IP_rotate_any_angle_ex. This new function allows you to specify the interpolation method for rotation and background color around the image being rotated. In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

Rotating the image multiple times at angles that are not multiple of 90 degrees may degrade the quality of the image.

You can only rotate PDF/PS images in 90 degree increments.

1.3.1.2.18.65 IG_IP_rotate_any_angle_bkgrnd

This function rotates the image by the specified angle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_any_angle_bkgrnd(
   HIGEAR hIGear,
   DOUBLE angle,
   AT_MODE rotate_mode,
   LPAT_PIXEL lpBkgrndColor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
angle	DOUBLE	Angle by which to rotate the image, in degrees. Positive values result in clockwise rotation; negative values result in counter-clockwise rotation.
rotate_mode	AT_MODE	Rotation mode. Specifies whether the image should be clipped or expanded. See enumIGRotationModes for possible values.
lpBkgrndColor	LPAT_PIXEL	A far pointer to the RGB or pixel value that specifies the background color to be used in the displaced areas after the image has been rotated.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;             // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;     // Count of errs on stack upon ret from func
AT_INT channelCount;     // Count of channels in the image
AT_PIXEL lpBackground[256];     // Buffer for background color
AT_INT i;

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get channel count
    IG_image_channel_count_get(hIGear, &channelCount);
    // Initialize background color with '255'
    for(i = 0; i < channelCount; i ++)
    {
        lpBackground[i] = (AT_PIXEL)255;
    }
    nErrcount = IG_IP_rotate_any_angle_bkgrnd(hIGear, 45., IG_ROTATE_CLIP, lpBackground);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The function rotates the image about its center point.

You can use IG_IP_rotate_compute_size to calculate the new dimensions of the bitmap that the image will have after rotation in IG_ROTATE_EXPAND mode.

The functionality of this API call has been upgraded and supported by the new function IG_IP_rotate_any_angle_ex. This new function allows you to specify the interpolation method for rotation. In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended.

Rotating the image multiple times at angles that are not multiple of 90 degrees may degrade the quality of the image.

You can only rotate PDF/PS images in 90 degree increments.

1.3.1.2.18.66 IG_IP_rotate_any_angle_ex

This function rotates the image by the specified angle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_any_angle_ex(
   HIGEAR hIGear,
   DOUBLE angle,
   AT_MODE rotate_mode,
   LPAT_PIXEL lpBkgrndColor,
   AT_MODE interpolation
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
angle	DOUBLE	Angle by which to rotate the image, in degrees. Positive values result in clockwise rotation; negative values result in counter-clockwise rotation.
rotate_mode	AT_MODE	Rotation mode. Specifies whether the image should be clipped or expanded. See enumIGRotationModes for possible values.
lpBkgrndColor	LPAT_PIXEL	A far pointer to the RGB or pixel value that specifies the background color to be used in the displaced areas after the image has been rotated.
interpolation	AT_MODE	Interpolation to use for rotation. Supported modes are IG_INTERPOLATION_NONE, IG_INTERPOLATION_BILINEAR, IG_INTERPOLATION_BICUBIC. Ignored for 1-bit images.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If interpolation is IG_INTERPOLATION_BILINEAR or IG_INTERPOLATION_BICUBIC:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.
- Images that have a Grayscale LUT attached to them.

Otherwise, all pixel formats supported by ImageGear Professional.

Interpolation mode is ignored for 1-bit images.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;     // Count of errs on stack upon ret from func
AT_INT channelCount;     // Count of channels in the image
AT_PIXEL lpBackground[256];     // Buffer for background color
AT_INT i;

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get channel count
    IG_image_channel_count_get(hIGear, &channelCount);
    // Initialize background color with '255'
    for(i = 0; i < channelCount; i ++)
    {
        lpBackground[i] = (AT_PIXEL)255;
    }
    nErrcount = IG_IP_rotate_any_angle_ex(hIGear, 45., IG_ROTATE_CLIP, lpBackground,
IG_INTERPOLATION_BILINEAR);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The function rotates the image about its center point.

You can use IG_IP_rotate_compute_size to calculate the new dimensions of the bitmap that the image will have after rotation in IG_ROTATE_EXPAND mode.

For the highest quality, bilinear interpolation is recommended, especially if the rotation angle is small (less than 5 degrees) and/or the image will be rotated multiple times. Bi-cubic interpolation can be used to achieve a slightly sharper appearance.

Rotating the image multiple times at angles that are not multiple of 90 degrees may degrade the quality of the image.

You can only rotate PDF/PS images in 90 degree increments.

1.3.1.2.18.67 IG_IP_rotate_compute_size

This function computes the new width and height of the image after it has been rotated using IG_IP_rotate_any_angle, IG_IP_rotate_any_angle_bkgrnd, or IG_IP_rotate_any_angle_ex functions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_compute_size(
   HIGEAR hIGear,
   DOUBLE angle,
   AT_MODE rotate_mode,
   LPAT_DIMENSION lpWidth,
   LPAT_DIMENSION lpHeight
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image of which to compute size.
angle	DOUBLE	Angle by which to rotate the image, in degrees. Positive values result in clockwise rotation; negative values result in counter-clockwise rotation.
rotate_mode	AT_MODE	Rotation mode. Specifies whether the image should be clipped or expanded. See enumIGRotationModes for possible values.
lpWidth	LPAT_DIMENSION	Pointer in which is returned the new width of the bitmap after rotation.
lpHeight	LPAT_DIMENSION	Pointer in which is returned the new height of the bitmap after rotation.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func
AT_DIMENSION nWidth, nHeight;   // New height and width of image


// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_rotate_compute_size(hIGear, 45., IG_ROTATE_EXPAND, &nWidth, &nHeight);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

You may use this function before rotation in IG_ROTATE_EXPAND mode to determine the dimensions of the new rotated image. This way, you can estimate the amount of memory that will be needed to hold the new image.

1.3.1.2.18.68 IG_IP_rotate_multiple_90

This function will rotate the image referenced by hIGear at an angle that is a multiple of 90 degrees.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_multiple_90(
   HIGEAR hIGear,
   AT_MODE nMult_90_Mode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to rotate.
nMult_90_Mode	AT_MODE	A constant that specifies rotation angle. See enumIGRotationValues

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_rotate_multiple_90(hIGear, IG_ROTATE_90);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The function rotates the image about its center point.

If the rotation is either 90 or 270 degrees, the previous width of the image becomes the new height, and the previous height of the image becomes the new width. IG_ROTATE_0 does nothing and is included for completeness only.

To rotate by an arbitrary angle that may not be a multiple of 90 degrees, use IG_IP_rotate_any_angle_ex.

The function does not swap vertical and horizontal DIB resolutions. If vertical and horizontal resolutions are different, rotation by 90 or 270 degrees will cause the image to display out of original proportions. Use IG_IP_rotate_multiple_90_opt function with SwapResolutions field of lpRotateOptions parameter set to TRUE to swap the resolutions during rotation and preserve the proportions.

1.3.1.2.18.69 IG_IP_rotate_multiple_90_opt

This function will rotate the image referenced by hIGear at an angle that is a multiple of 90 degrees, using additional rotation options.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_rotate_multiple_90_opt(
   HIGEAR hIGear,
   AT_MODE nMult_90_Mode,
   LPAT_ROTATE_MULTIPLE_90_OPTIONS lpRotateOptions
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to rotate.
nMult_90_Mode	AT_MODE	A constant that specifies rotation angle. See enumIGRotationValues.
lpRotateOptions	LPAT_ROTATE_MULTIPLE_90_OPTIONS	Pointer to AT_ROTATE_MULTIPLE_90_OPTIONS structure, which contains additional rotation options. NULL means the same behavior as IG_IP_rotate_multiple_90.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;     // Count of errs on stack upon ret from func
AT_ROTATE_MULTIPLE_90_OPTIONS rotateOptions = {TRUE}; // Rotation options

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_rotate_multiple_90_opt(hIGear, IG_ROTATE_90, &rotateOptions);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

The function rotates the image about its center point, using additional options.

If the rotation is either 90 or 270 degrees, the previous width of the image becomes the new height, and the previous height of the image becomes the new width. IG_ROTATE_0 does nothing and is included for completeness only.

To rotate by an arbitrary angle that may not be a multiple of 90 degrees, use IG_IP_rotate_any_angle_ex.

1.3.1.2.18.70 IG_IP_sharpen

This function sharpens the image by making the dark side of a contrast edge become darker and the light side lighter.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_sharpen(
   HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const INT nSharpFactor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Pointer to an AT_RECT struct specifying a portion of the image to be affected. NULL means entire image.
nSharpFactor	const INT	Factor indicating the degree for increasing image sharpness. Valid range is 1 to 5. The higher the value, the more sharpening will be applied.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_sharpen(hIGear, NULL, 2);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

Flat areas (areas that are filled by the same pixel value) are not altered by this function.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal NRA flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask.

See Also

IG_IP_unsharp_mask

IG_IP_smooth

IG_IP_convolve_matrix

1.3.1.2.18.71 IG_IP_smooth

This function makes images look softer and slightly out of focus.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_smooth(
   HIGEAR hIGear,
   const LPAT_RECT lpRect,
   const INT nSmoothFactor
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Pointer to an AT_RECT struct specifying a portion of the image to be affected. NULL means entire image.
nSmoothFactor	const INT	Factor indicating the degree of smoothness wanted. Valid range is 1 to 4. Larger values cause greater softening.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_smooth(hIGear, NULL, 2);
    // ...
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

This function can be used to reduce the graininess of an image.

The pixel neighborhood considered by this function is 3x3, 5x5, 7x7, or 9x9 for nSmoothFactor = 1, 2, 3, or 4 respectively. Therefore, smaller values result in faster processing, while larger values result in more smoothing.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal NRA flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask.

See Also

IG_IP_gaussian_blur

IG_FX_blur

IG_IP_sharpen

IG_IP_convolve_matrix

1.3.1.2.18.72 IG_IP_swap_red_blue

This function reverses the color sequence in the pixels of image hIGear's image bitmap.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_swap_red_blue ( HIGEAR hIGear);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Non-RGB images are converted to RGB for processing, and then back to original color space.

Example:

```
HIGEAR hIGear;       /* HIGEAR handle of 24-bit image */
IG_IP_swap_red_blue ( hIGear );
```

Remarks:

If the sequence is Blue-Green-Red (the standard sequence for a DIB), it is reversed to Red-Green-Blue. If Red-Green-Blue, each pixel is reversed to Blue-Green-Red. This function is typically used on 24-bit RGB images, but it can operate on other image types as well.

1.3.1.2.18.73 IG_IP_thumbnail_create

This function creates a resized copy of the image. It can be used for creating a thumbnail (small preview version of the image).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_thumbnail_create(
   HIGEAR hOriginalImage,
   LPHIGEAR lphNewThumbnail,
   AT_DIMENSION nNewWidth,
   AT_DIMENSION nNewHeight,
   AT_MODE nInterpMethod
);
```

Arguments:

Name	Type	Description
hOriginalImage	HIGEAR	HIGEAR handle of image of which to create thumbnail image.
lphNewThumbnail	LPHIGEAR	Pointer to a variable of type HIGEAR to receive the HIGEAR handle of the created thumbnail image.
nNewWidth	AT_DIMENSION	Specifies width wanted for the new image.
nNewHeight	AT_DIMENSION	Specifies height wanted for the new image.
nInterpMethod	AT_MODE	Specifies interpolation method to use for image resizing. See enumIGInterpolations for possible values.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If nInterpMethod is IG_INTERPOLATION_GRAYSCALE, IG_INTERPOLATION_PRESERVE_WHITE, or IG_INTERPOLATION_PRESERVE_BLACK:

- Indexed RGB - 1 bpp;
- Grayscale - 1 bpp.

If nInterpMethod is IG_INTERPOLATION_AVERAGE or IG_INTERPOLATION_BILINEAR:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.

If nInterpMethod is IG_INTERPOLATION_BICUBIC:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.
- Grayscale - 1 bpp.

Otherwise, all pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
HIGEAR hIGearThumb;          // HIGEAR handle of the thumbnail image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.bmp" from working directory
nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
```

```
{
    nErrcount = IG_IP_thumbnail_create(hIGear, &hIGearThumb, 64, 64,
IG_INTERPOLATION_BILINEAR);
    if(nErrcount == 0)
    {
        // ...
        // Destroy the thumbnail image
        IG_image_delete(hIGearThumb);
    }
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

This function works in the same way as IG_IP_resize except that it returns a resized copy of the original image, instead of changing the original image. See IG_IP_resize for additional details.

The functionality of this API call has been upgraded and supported by the new function IG_IP_thumbnail_create_ex. This new function allows you to pass additional parameters that affect interpolation.

1.3.1.2.18.74 IG_IP_thumbnail_create_ex

This function creates a resized copy of the image. It can be used for creating a thumbnail (small preview version of the image).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_thumbnail_create_ex(
   HIGEAR hOriginalImage,
   LPHIGEAR lphNewThumbnail,
   AT_DIMENSION nNewWidth,
   AT_DIMENSION nNewHeight,
   AT_MODE nInterpMethod,
   DWORD dwFlags,
   INT nValue
);
```

Arguments:

Name	Type	Description
hOriginalImage	HIGEAR	HIGEAR handle of image of which to create thumbnail image.
lphNewThumbnail	LPHIGEAR	Pointer to a variable of type HIGEAR to receive the HIGEAR handle of the created thumbnail image.
nNewWidth	AT_DIMENSION	Specifies width wanted for the new image.
nNewHeight	AT_DIMENSION	Specifies height wanted for the new image.
nInterpMethod	AT_MODE	Specifies interpolation method to use for image resizing. See enumIGInterpolations for possible values.
dwFlags	DWORD	The contents of this parameter depends upon the value of nInterpMethod. This is currently not used.
nValue	INT	The contents of this parameter depends upon the value of nInterpMethod: • IG_INTERPOLATION_GRAYSCALE - nValue can be from 0 to 100. It takes the proportion of pixels from entry 1 to entry 0 (white/black). • IG_INTERPOLATION_PRESERVE_WHITE - nValue can be from 0 to 100. It indicates the threshold value of the amount of white color to include. • IG_INTERPOLATION_PRESERVE_BLACK - nValue can be from 0 to 100. It indicates the threshold value of the amount of black color to include. • Any other values - nValue is ignored.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If nInterpMethod is IG_INTERPOLATION_GRAYSCALE, IG_INTERPOLATION_PRESERVE_WHITE, or IG_INTERPOLATION_PRESERVE_BLACK:

- Indexed RGB - 1 bpp;
- Grayscale - 1 bpp.

If nInterpMethod is IG_INTERPOLATION_AVERAGE or IG_INTERPOLATION_BILINEAR:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.

If nInterpMethod is IG_INTERPOLATION_BICUBIC:

All pixel formats supported by ImageGear Professional, except:

- Indexed RGB with non-grayscale palette.

- Grayscale - 1 bpp.

Otherwise, all pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;              // HIGEAR handle of the image
HIGEAR hIGearThumb;          // HIGEAR handle of the thumbnail image
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func

// Load image file "picture.tif", 1 bpp, from working directory
nErrcount = IG_load_file("picture.tif", &hIGear);
if(nErrcount == 0)
{
    nErrcount = IG_IP_thumbnail_create_ex(hIGear, &hIGearThumb, 64, 64,
IG_INTERPOLATION_GRAYSCALE, 0, 50);
    if(nErrcount == 0)
    {
        // ...
        // Destroy the thumbnail image
        IG_image_delete(hIGearThumb);
    }
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

This function works in the same way as IG_IP_resize_ex with the only difference that it returns a resized copy of the original image, instead of changing the original image. See IG_IP_resize_ex for additional details.

1.3.1.2.18.75 IG_IP_transform_with_LUT

This function transforms the pixel values of the image referenced by hIGear, using a LUT.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_transform_with_LUT (
        HIGEAR hIGear,
        LPAT_RECT lpRect,
        LPAT_PIXEL lpLUTr,
        LPAT_PIXEL lpLUTg,
        LPAT_PIXEL lpLUTb,
        AT_MODE nColorMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to transform.
lpRect	LPAT_RECT	Rectangular portion of the image to process; set to NULL for entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
lpLUTr	LPAT_PIXEL	Far pointer to a user-supplied Look-Up Table for transforming the red component, or for transforming the pixel value if the image is less than 24-bit.
lpLUTg	LPAT_PIXEL	Far pointer to a user-supplied Look-Up Table for transforming the green component, or for transforming the pixel value if the image is 24-bit.
lpLUTb	LPAT_PIXEL	Far pointer to a user-supplied Look-Up Table for transforming the blue component, or for transforming the pixel value if the image is 24-bit.
nColorMode	AT_MODE	A variable of type AT_MODE (IG_COLOR_COMP_) that tells which color channel to use, or to use all three.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, with the following restrictions:
Bits per channel must be less than or equal to 8.

Example:

```
HIGEAR hIGear;          /* HIGEAR handle of image    */
AT_PIXEL LUTred[256];   /* Look up pixels, or red if 24-bit, here */
AT_PIXEL LUTgreen[256];/* Look up green if 24-bit, here    */
AT_PIXEL LUTblue[256]; /* Look up blue if 24-bit, here     */
IG_IP_transform_with_LUT ( hIGear, (LPAT_PIXEL)&LUTred,(LPAT_PIXEL)&LUTgreen,
(LPAT_PIXEL)&LUTblue, IG_COLOR_COMP_RGB );
```

Remarks:

The pixels from hIGear are used as indices into the LUT. The entry in the LUT at this position is placed into the new image. For 24-bit images, the three channels each have their own LUT. You can point all three LUT parameters to the same LUT. This will process all three channels the same. For 8-bit gray level images, only the lpLUTr LUT parameter is used and the pixel value replaced from the LUT is as follows and the other two are ignored.

```
new pixel value  = RedLUT[ old pixel value ].
```

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. (See above.) However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

Please see the descriptions of IG_IP_NR_ROI_mask_associate() and IG_IP_NR_ROI_to_HIGEAR_mask() functions for more details.

1.3.1.2.18.76 IG_IP_transform_with_LUT_ex

This function transforms the pixel values of the image referenced by hIGear, using a LUT.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_transform_with_LUT_ex(
        HIGEAR hImage,
        LPAT_RECT lpRect,
        LPAT_VOID* lpLUTs,
        AT_INT nLUTNumber,
        LPAT_INT lpLUTSize,
        AT_MODE nChannels
);
```

Arguments:

Name	Type	Description
hImage	HIGEAR	Image to transform.
lpRect	LPAT_RECT	Rectangular portion of the image to process; set to NULL for entire image. Before ImageGear performs this operation, it will check to see if an internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
lpLUTs	LPAT_VOID*	Far pointer to a user-supplied Look-Up Table.
nLUTNumber	AT_INT	Number of LUTs in lpLUTs.
lpLUTSize	LPAT_INT	Size of every LUT.
nChannels	AT_MODE	A variable of type AT_MODE (IG_COLOR_COMP_) that tells which color channel to use.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, with the following restrictions:
Bits per channel must be less than or equal to 16.

Remarks:

The pixels from hIGear are used as indices into the LUT. The entry in the LUT at this position is placed into the new image. The function supports images having any number of channels and arbitrary channel depths.

This function, like other ImageGear Image Processing and Clipboard API calls, takes an AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

1.3.1.2.18.77 IG_IP_unsharp_mask

Unsharp masking filter is used for image sharpening and edge enhancement.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_IP_unsharp_mask (
        HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const double dblRadius,
        const UINT nAmount,
        const UINT nThreshold,
        const AT_MODE nColorChannel );
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRect	const LPAT_RECT	Far pointer to AT_RECT struct specifying a portion of the image to be affected. NULL means entire image. Before ImageGear performs this operation, it will check to see if internal flag has been set to TRUE to make a mask active for this HIGEAR image. If a mask is active, and a valid pointer to a mask can be found, ImageGear will override the settings passed to this structure in favor of the non-rectangular ROI defined by the mask.
dblRadius	const double	Defines the neighborhood to be considered for each pixel.
nAmount	const UINT	Amount of sharpening, in percents.
nThreshold	const UINT	Minimal difference between a pixel and its neighbors at which the pixel will be modified.
nColorChannel	const AT_MODE	Color channel to which the transform shall be applied.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional, except:
Indexed RGB with non-grayscale palette.

Example:

```
HIGEAR hIGear;
/* Sharpen whole image, radius = 2.0, amount = 150, threshold = 10 */
nErrCount =IG_IP_unsharp_mask ( hIGear, NULL, 2.0, 150, 10, IG_COLOR_CHANNEL_ALL );
```

Remarks:

The algorithm works by subtracting a smoothed version of the image ("unsharp") from the original image.

Parameter dblRadius can range from 0.1 to 500. Typical values for high-resolution images range from 1.0 to 2.0. Use larger values of dblRadius for thicker edges. The smaller dblRadius value is, the faster is this processing .

Parameter nAmount can range from 1 to 500. Typical values for photographic images range from 100 to 200. Use bigger values for greater edge contrast.

Parameter nThreshold can range from 1 to 500. Typical values for photographic images are between 0 and 20. Use nThreshold to apply sharpening only to those areas where contrast changes significantly. Flat areas will remain unchanged. A proper selection of nThreshold allows you to enhance edges on the image while leaving insufficient details, such as noise or grain of the photographic film, unchanged.

Parameter nColorChannel specifies the channel to which the transform shall be applied. The default value is IG_COLOR_COMP_ALL: apply transform to all color channels of the image. To run UnsharpMask process on the Intensity channel of the image, convert the image to YUV colorspace, run UnsharpMask filter with nColorChannel = IG_COLOR_COMP_YUV_Y, and convert back to original colorspace.

This function, like other ImageGear Image Processing and Clipboard API calls, takes AT_RECT structure as an argument, so that you can process a rectangular sub-region of an image. However, before ImageGear performs the operation specified by this function, it will check to see if an internal flag has been set to TRUE, indicating that a mask HIGEAR should be used with the image. If the flag is set to TRUE, and a valid pointer to a mask image has been assigned, ImageGear will override the settings passed to the AT_RECT structure and use the non-rectangular ROI defined by the mask HIGEAR. To create a non-rectangular region of interest, call IG_IP_NR_ROI_to_HIGEAR_mask().

1.3.1.2.19 Info Functions

This section provides information about the Info group of functions.

- IG_info_get
- IG_info_get_ex
- IG_info_get_FD
- IG_info_get_FD_ex
- IG_info_get_mem
- IG_info_get_mem_ex
- IG_page_count_get
- IG_page_count_get_FD
- IG_page_count_get_mem
- IG_tile_count_get
- IG_tile_count_get_FD
- IG_tile_count_get_mem

1.3.1.2.19.1 IG_info_get

This function obtains information about the specified file without loading the pixel data. This is an obsolete function, see remarks.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_info_get(
   const LPSTR lpszFileName,
   LPAT_MODE lpFileType,
   LPAT_MODE lpCompression,
   LPAT_DIB lpDIB
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Name of file about which to get information.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values.
lpCompression	LPAT_MODE	Pointer to an AT_MODE variable in which compression type will be returned. See enumIGCompressions for possible values.
lpDIB	LPAT_DIB	Pointer to an AT_DIB structure to which additional file information such as width, height, and Bits Per Pixel will be returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If lpDIB is not NULL, then

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

otherwise, all pixel formats supported by ImageGear Professional.

Example:

```
AT_MODE nFileType;          // Will receive an IG_FORMAT_ constant
AT_MODE nCompression;      // Will receive an IG_COMPRESSION_    constant
AT_DIB dibInfoDIB;          // Will receive copy of the    BITMAPINFOHEADER
AT_ERRCOUNT nErrcount;      // Returned count of errors
nErrcount = IG_info_get("picture.bmp", &nFileType, &nCompression, &dibInfoDIB);
```

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_info_get_ex instead.

Any of the output parameters such as lpFileType, lpCompression or lpDIB can be NULL, if the corresponding info is not required.

1.3.1.2.19.2 IG_info_get_ex

This function obtains information about the specified file page, without loading the pixel data.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_info_get_ex(
   const LPSTR lpszFileName,
   UINT nPageNumber,
   LPAT_MODE lpFileType,
   LPAT_MODE lpCompression,
   HIGDIBINFO* lphDIB
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Path and name of the file to get the information about. The path can be absolute or relative.
nPageNumber	UINT	Page number to get info about if this is a multi-page (multi-image) file. Note that page numbers begin at 1, not 0. Set nPageNumber to 1, if this is not a multi-page file.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values.
lpCompression	LPAT_MODE	Pointer to an AT_MODE variable in which compression type will be returned. See enumIGCompressions for possible values.
lphDIB	HIGDIBINFO*	Pointer to an HIGDIBINFO structure to which additional file information such as width, height, and Bits Per Pixel, will be returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_MODE nFileType;         // Will receive an IG_FORMAT_ constant
AT_MODE nCompression;     // Will receive an IG_COMPRESSION_ constant
HIGDIBINFO hDIB;
AT_ERRCOUNT nErrcount;         // Returned count of errors
nErrcount = IG_info_get_ex("picture.bmp", 1, &nFileType, &nCompression, &hDIB);

// ...
// Delete DIB info
IG_DIB_info_delete(hDIB);
```

Remarks:

Any of the output parameters such as lpFileType, lpCompression or lphDIB can be NULL, if the corresponding info is not required.

See also the section Getting Information and Sorting Images.

This function is identical to IG_fltr_pageinfo_get_ex.

1.3.1.2.19.3 IG_info_get_FD

This function obtains information about the file specified by the file handle, without loading the pixel data. This is an obsolete function, see remarks.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_info_get_FD(
   AT_INT fd,
   LONG lOffset,
   UINT nPage,
   LPAT_MODE lpFileType,
   LPAT_MODE lpCompression,
   LPAT_DIB lpDIB
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file. This handle can be obtained from Microsoft Windows function such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. lOffset is usually 0.
nPage	UINT	Page number for which the info is obtained. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values.
lpCompression	LPAT_MODE	Pointer to an AT_MODE variable in which compression type will be returned. See enumIGCompressions for possible values.
lpDIB	LPAT_DIB	Pointer to an AT_DIB structure to which other file information such as width, height, and Bits Per Pixel will be returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If lpDIB is not NULL, then

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

else - all pixel formats supported by ImageGear Professional.

Example:

```
HANDLE fd;                  // File Descriptor
AT_MODE nFileType;          // Will receive an IG_FORMAT_ constant
AT_MODE nCompression;     // Will receive an IG_COMPRESSION_ constant
AT_DIB atDIB;
AT_ERRCOUNT nErrcount;         // Returned count of errors

fd = CreateFile(_T("picture.bmp"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
```

```
if(fd != INVALID_HANDLE_VALUE)
{
    nErrcount = IG_info_get_FD((AT_INT)fd, 0, 1, &nFileType, &nCompression, &atDIB);
    CloseHandle(fd);

    // ...
}
```

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_info_get_FD_ex instead.

Any of the output parameters such as lpFileType, lpCompression or lpDIB can be NULL, if the corresponding info is not required.

1.3.1.2.19.4 IG_info_get_FD_ex

This function obtains information about the file specified by the file handle, without loading its pixel data.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_info_get_FD_ex(
   AT_INT fd,
   LONG lOffset,
   UINT nPage,
   LPAT_MODE lpFileType,
   LPAT_MODE lpCompression,
   HIGDIBINFO* lphDIB
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file. This handle can be obtained from Microsoft Windows function such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. lOffset is usually 0.
nPage	UINT	Page number about which to get information, if a multi-page file set to 1 or greater; for a non-multi-page file set to 1.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values.
lpCompression	LPAT_MODE	Pointer to an AT_MODE variable in which compression type will be returned. See enumIGCompressions for possible values.
lphDIB	HIGDIBINFO*	Pointer to an HIGDIBINFO structure to which other file information such as width, height, and Bits Per Pixel, will be returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HANDLE fd;                // File Descriptor
AT_MODE nFileType;        // Will receive an IG_FORMAT_ constant
AT_MODE nCompression;    // Will receive an IG_COMPRESSION_ constant
HIGDIBINFO hDIB;
AT_ERRCOUNT nErrcount;         // Returned count of errors

fd = CreateFile(_T("picture.bmp"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if(fd != INVALID_HANDLE_VALUE)
{
    nErrcount = IG_info_get_FD_ex((AT_INT)fd, 0, 1, &nFileType, &nCompression, &hDIB);
    CloseHandle(fd);

    // ...
    // Delete DIB info
    IG_DIB_info_delete(hDIB);
}
```

See Also

IG_info_get_ex

1.3.1.2.19.5 IG_info_get_mem

This function obtains information about the image located in a memory buffer. This is an obsolete function, see remarks.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_info_get_mem(
   LPVOID lpImage,
   AT_UINT nImageSize,
   UINT nPage,
   LPAT_MODE lpFileType,
   LPAT_MODE lpCompression,
   LPAT_DIB lpDIB
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to memory location of an image file that is currently in memory.
nImageSize	AT_UINT	Size of image in memory.
nPage	UINT	Page number for which the info is obtained. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page memory file.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values.
lpCompression	LPAT_MODE	Pointer to an AT_MODE variable in which compression type will be returned. See enumIGCompressions for possible values.
lpDIB	LPAT_DIB	Pointer to an AT_DIB structure to which other file information such as width, height, and Bits Per Pixel will be returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

If lpDIB is not NULL, then

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

otherwise, all pixel formats supported by ImageGear Professional.

Example:

```
char* lpBuffer;              // Memory buffer with the image
AT_UINT nBufferSize;     // Size of the memory buffer
AT_MODE nFileType;         // Will receive an IG_FORMAT_ constant
AT_MODE nCompression;     // Will receive an IG_COMPRESSION_ constant
AT_DIB atDIB;
AT_ERRCOUNT nErrcount;         // Returned count of errors

// Open a file and get its size
FILE* fp = NULL;
fopen_s(&fp, "picture.bmp", "rb");
if(fp != NULL)
```

```
{
    fseek(fp, 0, SEEK_END);
    nBufferSize = (AT_UINT)ftell(fp);
    fseek(fp, 0, SEEK_SET);
    // Allocate memory and read the image into the memory buffer
    lpBuffer = (char*)malloc(nBufferSize);
    fread(lpBuffer, 1, nBufferSize, fp);
    // File is no longer needed - close it
    fclose(fp);
    // Get image info
    nErrcount = IG_info_get_mem(lpBuffer, nBufferSize, 1, &nFileType, &nCompression,
&atDIB);
    fclose(fp);

    // ...
    // Delete memory buffer
    free(lpBuffer);
}
```

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_info_get_mem_ex instead.

Any of the output parameters such as lpFileType, lpCompression or lpDIB can be NULL, if the corresponding info is not required.

1.3.1.2.19.6 IG_info_get_mem_ex

This function obtains information about the image located in the memory buffer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_info_get_mem_ex(
   VOID FAR32* lpImage32,
   AT_UINT dwSize,
   UINT nPage,
   LPAT_MODE lpFileType,
   LPAT_MODE lpCompression,
   HIGDIBINFO* lphDIB
);
```

Arguments:

Name	Type	Description
lpImage32	VOID FAR32*	Pointer to start of file image in memory.
dwSize	AT_UINT	Size of image in memory.
nPage	UINT	Page number about which to get information, if a multi page file set to 1 or greater; for a non multi page file set to 1.
lpFileType	LPAT_MODE	Pointer to an AT_MODE variable in which the file type will be returned. See enumIGFormats for possible values.
lpCompression	LPAT_MODE	Pointer to an AT_MODE variable in which the compression type will be returned. See enumIGCompressions for possible values.
lphDIB	HIGDIBINFO*	Pointer to an HIGDIBINFO structure to which other file information such as width, height, and Bits Per Pixel, will be returned.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
char* lpBuffer;              // Memory buffer with the image
AT_UINT nBufferSize;    // Size of the memory buffer
AT_MODE nFileType;        // Will receive an IG_FORMAT_ constant
AT_MODE nCompression;    // Will receive an IG_COMPRESSION_ constant
HIGDIBINFO hDIB;
AT_ERRCOUNT nErrcount;         // Returned count of errors

// Open a file and get its size
FILE* fp = NULL;
fopen_s(&fp, "picture.bmp", "rb");
if(fp != NULL)
{
    fseek(fp, 0, SEEK_END);
    nBufferSize = (AT_UINT)ftell(fp);
    fseek(fp, 0, SEEK_SET);
    // Allocate memory and read the image into the memory buffer
    lpBuffer = (char*)malloc(nBufferSize);
    fread(lpBuffer, 1, nBufferSize, fp);
    // File is no longer needed - close it
    fclose(fp);
    // Get image info
    nErrcount = IG_info_get_mem_ex(lpBuffer, nBufferSize, 1, &nFileType, &nCompression, 
&hDIB);
    fclose(fp);

    // ...
    // Delete memory buffer
    free(lpBuffer);
    // Delete DIB info
    IG_DIB_info_delete(hDIB);
}
```

Remarks:

See also IG_info_get_FD_ex and IG_info_get_ex functions.

1.3.1.2.19.7 IG_page_count_get

This function obtains the number of pages in the image file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_page_count_get(
   const LPSTR lpszFileName,
   LPUINT lpPageCount
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Path and name of the file for which to get the page count. The path can be absolute or relative.
lpPageCount	LPUINT	Pointer to a UINT variable to receive page count.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
UINT nPages;             // Will receive number of pages
AT_ERRCOUNT nErrcount;   // Count of returned errors
nErrcount = IG_page_count_get ( "picture.bmp", &nPages );
```

See Also

IG_fltr_pagecount_file_format

1.3.1.2.19.8 IG_page_count_get_FD

This function obtains the number of pages in the image file specified by its file handle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_page_count_get_FD(
   AT_INT fd,
   LONG lOffset,
   LPUINT lpPageCount
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file. This handle can be obtained from Microsoft Windows function such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. lOffset is usually 0.
lpPageCount	LPUINT	Pointer to a UINT variable to receive page count .

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HANDLE fd;                  // File Descriptor
UINT nPages;              // Will receive number of pages
AT_ERRCOUNT nErrcount;    // Count of returned errors

fd = CreateFile(_T("picture.bmp"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if(fd != INVALID_HANDLE_VALUE)
{
    nErrcount = IG_page_count_get_FD((AT_INT)fd, 0, &nPages);
    CloseHandle(fd);
}
```

Remarks:

Call this function when the file is already opened and you have its File Descriptor handle (fd).

See Also

IG_fltr_pagecount_FD_format

1.3.1.2.19.9 IG_page_count_get_mem

This function obtains the number of pages in the memory image file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_page_count_get_mem(
   LPVOID lpImage,
   AT_UINT nImageSize,
   LPUINT lpPageCount
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to start of file image in memory.
nImageSize	AT_UINT	Size of image in memory.
lpPageCount	LPUINT	Pointer to a UINT variable to receive page count.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
char* lpBuffer;             // Memory buffer with the image
AT_UINT nBufferSize;    // Size of the memory buffer
UINT nPageCount;        // Will receive number of pages
AT_ERRCOUNT nErrcount;        // Returned count of errors

// Open a file and get its size
FILE* fp = NULL;
fopen_s(&fp, "picture.bmp", "rb");
if(fp != NULL)
{
    fseek(fp, 0, SEEK_END);
    nBufferSize = (AT_UINT)ftell(fp);
    fseek(fp, 0, SEEK_SET);
    // Allocate memory and read the image into the memory buffer
    lpBuffer = (char*)malloc(nBufferSize);
    fread(lpBuffer, 1, nBufferSize, fp);
    // File is no longer needed - close it
    fclose(fp);
    // Get image info
    nErrcount = IG_page_count_get_mem(lpBuffer, nBufferSize, &nPageCount);
    fclose(fp);

    // ...
    // Delete memory buffer
    free(lpBuffer);
}
```

Remarks:

See also functions IG_page_count_get_FD and IG_page_count_get.

1.3.1.2.19.10 IG_tile_count_get

This function gets the number of tiles constituting a page for file formats that support tiled pages.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_tile_count_get(
   const LPSTR lpszFileName,
   UINT nPageNum,
   LPUINT lpTileCountH,
   LPUINT lpTileCountV
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Path and name of the file for which to get the tile count. The path can be absolute or relative.
nPageNum	UINT	Page number for which to get the count of tiles.
lpTileCountH	LPUINT	Pointer to a UINT variable to receive the number of tiles horizontally (number of tiles in a row).
lpTileCountV	LPUINT	Pointer to a UINT variable to receive the number of tiles vertically (number of tiles in a column).

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
UINT nTileRows;             // Will receive number of tile rows
UINT nTileCols;             // Will receive number of tile cols
AT_ERRCOUNT nErrcount;  // Returned count of errors
// Get number of tiles, first page of file:
nErrcount = IG_tile_count_get("picture_tiled.tif", 1, &nTileRows, &nTileCols );
```

Remarks:

The function returns 0 for both lpTileCountH and lpTileCountV if the image file format does not support tiled images.

1.3.1.2.19.11 IG_tile_count_get_FD

This function gets the number of tiles constituting a page for file formats that support tiled pages in the image file specified by its file handle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_tile_count_get_FD(
   AT_INT fd,
   LONG lOffset,
   UINT nPageNum,
   LPUINT lpTileCountH,
   LPUINT lpTileCountV
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file. This handle can be obtained from Microsoft Windows function such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. lOffset is usually 0.
nPageNum	UINT	Page number for which to get count of tiles.
lpTileCountH	LPUINT	Pointer to a UINT variable to receive the number of tiles horizontally (number of tiles in a row).
lpTileCountV	LPUINT	Pointer to a UINT variable to receive the number of tiles vertically (number of tiles in a column).

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HANDLE fd;                 // File Descriptor
UINT nTileRows;             // Will receive number of tile rows
UINT nTileCols;             // Will receive number of tile cols
AT_ERRCOUNT nErrcount;  // Count of returned errors

fd = CreateFile(_T("picture_tiled.tif"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if(fd != INVALID_HANDLE_VALUE)
{
    nErrcount = IG_tile_count_get_FD((AT_INT)fd, 0, 1, &nTileRows, &nTileCols );
    CloseHandle(fd);
}
```

Remarks:

Use this function when the file is open and you have its file handle.

The function returns 0 for both lpTileCountH and lpTileCountV if the image file format does not support tiled images.

1.3.1.2.19.12 IG_tile_count_get_mem

This function gets the number of tiles constituting a page for file formats that support tiled pages for the image files located in the memory buffer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_tile_count_get_mem(
   LPVOID lpImage,
   AT_UINT nImageSize,
   UINT nPageNum,
   LPUINT lpTileCountH,
   LPUINT lpTileCountV
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to the start of the image file in memory.
nImageSize	AT_UINT	Size of image file in memory.
nPageNum	UINT	Page number for which to get the count of tiles.
lpTileCountH	LPUINT	Pointer to a UINT variable to receive the number of tiles horizontally (number of tiles in a row).
lpTileCountV	LPUINT	Pointer to a UINT variable to receive the number of tiles vertically (number of tiles in a column).

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
char* lpBuffer;              // Memory buffer with the image
AT_UINT nBufferSize;      // Size of the memory buffer
UINT nTileRows;              // Will receive number of tile rows
UINT nTileCols;              // Will receive number of tile cols
AT_ERRCOUNT nErrcount;          // Returned count of errors

// Open a file and get its size
FILE* fp = NULL;
fopen_s(&fp, "picture_tiled.tif", "rb");
if(fp != NULL)
{
    fseek(fp, 0, SEEK_END);
    nBufferSize = (AT_UINT)ftell(fp);
    fseek(fp, 0, SEEK_SET);
    // Allocate memory and read the image into the memory buffer
    lpBuffer = (char*)malloc(nBufferSize);
    fread(lpBuffer, 1, nBufferSize, fp);
    // File is no longer needed - close it
    fclose(fp);
    // Get image info
    nErrcount = IG_tile_count_get_mem(lpBuffer, nBufferSize, 1, &nTileRows, &nTileCols );
    fclose(fp);
    // ...
    // Delete memory buffer
    free(lpBuffer);
}
```

Remarks:

Use this function when the file image is in memory.

The function returns 0 for both lpTileCountH and lpTileCountV if the image file format does not support tiled images.

1.3.1.2.20 Licensing Functions

This section provides information about the Licensing group of functions.

- IG_lic_OEM_license_key_set
- IG_lic_solution_key_set
- IG_lic_solution_name_set

1.3.1.2.20.1 IG_lic_OEM_license_key_set

This function specifies the License Key to ImageGear for the deployment licensing model.

Declaration:

```
AT_ERRCODE  ACCUAPI  IG_lic_OEM_license_key_set( LPCHAR lpLicenseKey );
```

Arguments:

Name	Type	Description
lpLicenseKey	LPCHAR	The ImageGear license key string.

Return Value:

Returns the result code of this function call.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.20.2 IG_lic_solution_key_set

This function specifies your deployment Solution Key for the deployment licensing models.

Declaration:

```
VOID ACCUAPI IG_lic_solution_key_set (
        DWORD dwKey1,
        DWORD dwKey2,
        DWORD dwKey3,
        DWORD dwKey4
);
```

Arguments:

Name	Type	Description
dwKey1	DWORD	Key1 component of the solution key.
dwKey2	DWORD	Key2 component of the solution key.
dwKey3	DWORD	Key3 component of the solution key.
dwKey4	DWORD	Key4 component of the solution key.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.20.3 IG_lic_solution_name_set

This function provides solution name for your ImageGear license for the deployment licensing models.

Declaration:

```
AT_ERRCODE ACCUAPI IG_lic_solution_name_set ( LPCHAR lpSolutionName);
```

Arguments:

Name	Type	Description
lpSolutionName	LPCHAR	The license solution name.

Return Value:

Returns the result code of this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Please see the section .

1.3.1.2.21 Load Functions

This section provides information about the Load group of functions.

- IG_load_alpha_mode_get
- IG_load_alpha_mode_set
- IG_load_auto_detect_get
- IG_load_auto_detect_set
- IG_load_CCITT_FD
- IG_load_CCITT_mem
- IG_load_color_reduction_get
- IG_load_color_reduction_set
- IG_load_extra_mode_get
- IG_load_extra_mode_set
- IG_load_FD
- IG_load_FD_CB
- IG_load_FD_CB_ex
- IG_load_file
- IG_load_file_display
- IG_load_mem
- IG_load_mem_CB
- IG_load_mem_CB_ex
- IG_load_raw_FD
- IG_load_raw_file
- IG_load_raw_mem
- IG_load_rect_get
- IG_load_rect_set
- IG_load_size_get
- IG_load_size_set
- IG_load_tag_CB_register
- IG_load_thumbnail
- IG_load_thumbnail_FD
- IG_load_thumbnail_mem
- IG_load_tiles_stitch
- IG_load_tiles_stitch_FD
- IG_load_tiles_stitch_mem

1.3.1.2.21.1 IG_load_alpha_mode_get

This function retrieves the last setting made by calling function IG_load_alpha_mode_set().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_alpha_mode_get( enumIGAlphaMode* lpMode);
```

Arguments:

Name	Type	Description
lpMode	enumIGAlphaMode*	Pointer to an enumIGAlphaMode variable to receive the current Alpha channel loading mode setting.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumIGAlphaMode AlphaMode;
IG_load_alpha_mode_get ( &AlphaMode );
```

1.3.1.2.21.2 IG_load_alpha_mode_set

This function instructs ImageGear to load or ignore alpha channel when loading an image that contains one.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_alpha_mode_set( enumIGAlphaMode Mode);
```

Arguments:

Name	Type	Description
Mode	enumIGAlphaMode	Alpha loading mode to be set. IG_ALPHA_MODE_KEEP (default) forces Alpha channel if it is present; IG_ALPHA_MODE_IGNORE ignores Alpha channel.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Ignore Alpha channel when loading: */
IG_load_alpha_mode_set (IG_ALPHA_MODE_IGNORE );
```

Remarks:

See also IG_load_alpha_mode_get function.

1.3.1.2.21.3 IG_load_auto_detect_get

This function obtains the current state of the format filter indicated by nFormatType.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_auto_detect_get(
   AT_MODE nFormatType,
   LPAT_BOOL lpToggle
);
```

Arguments:

Name	Type	Description
nFormatType	AT_MODE	A constant indicating the format filter for which the detection setting should be obtained. See enumIGFormats for possible values.
lpToggle	LPAT_BOOL	Pointer to a variable of type AT_BOOL in which will be returned the current state of the filter: enabled (TRUE), or disabled (FALSE).

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
BOOL      bEnabled; /* Will be TRUE if this file format can be accessed */
AT_ERRCOUNT nErrCount = IG_load_auto_detect_get ( IG_FORMAT_TIF , &bEnabled );
```

Remarks:

By default, detection is enabled for all ImageGear file format filters, except TXT (ASCII).

See also function IG_load_auto_detect_set.

1.3.1.2.21.4 IG_load_auto_detect_set

This function enables or disables a format filter.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_auto_detect_set(
   AT_MODE nFormatType,
   AT_BOOL bToggle
);
```

Arguments:

Name	Type	Description
nFormatType	AT_MODE	A constant indicating the format filter for which the detection setting should be modified. See enumIGFormats for possible values.
bToggle	AT_BOOL	Set to TRUE to enable detection; set to FALSE to disable detection. The default value is TRUE for most ImageGear format filters.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
BOOL bEnabled = FALSE; /* If FALSE this file format will be disabled */
AT_ERRCOUNT nErrCount = IG_load_auto_detect_set ( IG_FORMAT_TIF , bEnabled);
```

Remarks:

If detection of a specific file format is disabled, that file format type cannot be accessed by the image info getting and loading functions, such as IG_info_get_ex or IG_fltr_load_file.

By default, detection is enabled for all ImageGear file format filters, except TXT (ASCII) .

See also function IG_load_auto_detect_get.

1.3.1.2.21.5 IG_load_CCITT_FD

This function has been deprecated and will be removed from the public API in a future release. Please use IG_load_raw_FD instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_CCITT_FD(
   AT_INT fd,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight,
   AT_MODE nType,
   AT_MODE nFillOrder,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file containing the image to be loaded. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
nWidth	AT_DIMENSION	Number of pixels in each row of data to be read.
nHeight	AT_DIMENSION	Number of rows of data.
nType	AT_MODE	Specifies the type of compression: IG_COMPRESSION_CCITT_G3, IG_COMPRESSION_CCITT_G4 or IG_COMPRESSION_G32D. There is no default for this property.
nFillOrder	AT_MODE	IG_FILL_MSB or IG_FILL_LSB, specifying whether the most-significant-bit-first or least-significant-bit-first. There is no default for this property.
lphIGear	LPHIGEAR	Pointer to HIGEAR variable to return a newly created image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1 bpp.

Example:

```
HIGEAR hIGear = 0;              // Will receive HIGEAR image handle
HANDLE fd;                      // File Descriptor handle

// Open a dile to read
fd = CreateFile(_T("Group4.raw"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

if(fd != INVALID_HANDLE_VALUE)
{
    AT_ERRCOUNT nErrcount;         /* Returned count of errors */
    nErrcount = IG_load_CCITT_FD((AT_INT)fd, 2320, 3408, IG_COMPRESSION_CCITT_G4,
IG_FILL_MSB,     &hIGear);

    //...

    // Destroy the image
    if(IG_image_is_valid(hIGear))
    {
        IG_image_delete(hIGear);
    }
    CloseHandle(fd);
}
```

Remarks:

This function creates an ImageGear image from a raw CCITT Compressed data file.

File pointer must be positioned at the start of the data. (For example, your application should read or seek past any header that is present.) You must specify the type, G3 or G4, by means of argument nType, and you must specify the fill order, most-significant-bit-first or least-significant-bit-first, by argument nFillOrder. The most common fill order is most-significant-bit-first or IG_FILL_MSB.

The width and height of the image are specified by nWidth and nHeight. The handle of the resulting new ImageGear image is returned in the HIGEAR variable pointed to by lphIGear. The resulting image is always 1-bit.

This function is used when you have a non-standard or proprietary G3 or G4 compressed image file and you know the details of the header. There are literally hundreds of different types of image files that fall into this category. In order to be able to successfully read an image of this type you must know enough about the header to find where the height and width are stored. You can usually look at the header with a hex dump utility and see where these values are stored. Once you are able to read past the header plus get the dimensions of the image, you can then use this function. You can experiment with the other settings until the image is read correctly. See also function IG_load_auto_detect_set.

The functionality of this API call has been upgraded and supported by the new function IG_load_raw_FD. The reason that this new function has been created to expand the number of raw image types you can load into ImageGear.

In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended. Consider using IG_load_raw_FD instead.

G3 compressed images are always 1728 pixels wide. Since G3 files usually have a special code at the end of the image that ImageGear will detect, you can set the height to a value greater than the expected height of the image and it will be corrected once the end of image marker is detected. For G4 files the height and width must be known and in all cases the file pointer must be at the start of the compressed image when this function is called.

1.3.1.2.21.6 IG_load_CCITT_mem

This function has been deprecated and will be removed from the public API in a future release. Please use IG_load_raw_mem instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_CCITT_mem(
   LPVOID lpImage,
   AT_UINT nSize,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight,
   AT_MODE nType,
   AT_MODE nFillOrder,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Memory buffer containing raw Fax data to be loaded.
nSize	AT_UINT	Size of the memory buffer lpImage.
nWidth	AT_DIMENSION	Number of pixels in each row of data to be read.
nHeight	AT_DIMENSION	Number of rows of data.
nType	AT_MODE	Specifies the type of compression: IG_COMPRESSION_CCITT_G3, IG_COMPRESSION_CCITT_G4 or IG_COMPRESSION_G32D. There is no default for this property.
nFillOrder	AT_MODE	IG_FILL_MSB or IG_FILL_LSB, specifying whether the most-significant-bit-first or least-significant-bit-first. There is no default for this property.
lphIGear	LPHIGEAR	Pointer to HIGEAR variable to return a newly created image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear = 0;             // Will receive HIGEAR image handle
AT_BYTE* lpImage;         // Memory buffer to keep an image
AT_INT fileSize;         // Size of the memory buffer
FILE* fd = NULL;         // File Descriptor
AT_ERRCOUNT nErrcount;   // Returned count of errors */

// Open a file and get its size
fopen_s(&fd, "Group4.raw", "rb");
if(fd != NULL)
{
    fseek(fd, 0, SEEK_END);
    fileSize = (AT_UINT)ftell(fd);
    fseek(fd, 0, SEEK_SET);
    // Allocate memory and read the image into the memory buffer
    lpImage = (AT_BYTE*)malloc(fileSize);
    fread(lpImage, 1, fileSize, fd);
    // File is no longer needed - close it
    fclose(fd);

    // Load image from the memory
    nErrcount = IG_load_CCITT_mem(lpImage, fileSize, 2320, 3408,
        IG_COMPRESSION_CCITT_G4, IG_FILL_MSB, &hIGear );
    // Delete memory buffer
    free(lpImage);

    //...

    // Destroy the image
    if(IG_image_is_valid(hIGear))
    {
        IG_image_delete(hIGear);
    }
}
```

Remarks:

This function creates an ImageGear image from raw CCITT Compressed data located in memory.

This function operates similarly to function IG_load_CCITT_FD. Note that lpImage must point to the start of the actual data (not to the start of any header information that may be present).

The functionality of this API call has been upgraded and supported by the new function IG_load_raw_mem. The reason that this new function has been created is that the old function restricted you to loading raw images that are stored with CCITT formatting.

In the interest of backward compatibility, we have left the old function in its original form and have retained support for it. If you have already used the old function in your code, it is not mandatory that you modify your code, but it is recommended. Consider using IG_load_raw_mem instead.

1.3.1.2.21.7 IG_load_color_reduction_get

This function retrieves the last setting made by calling function IG_load_color_reduction_set().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_color_reduction_get(
   LPUINT lpColorReduceMode
);
```

Arguments:

Name	Type	Description
lpColorReduceMode	LPUINT	Pointer to an AT_MODE variable to receive the current IG_LOAD_COLOR setting.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
UINT nLoadReduceMode;    /* Will receive the IG_LOAD_COLOR_ constant */
AT_ERRCOUNT nErrCount = IG_load_color_reduction_get ( &nLoadReduceMode );
```

Remarks:

If no color reduction is in effect, IG_LOAD_COLOR_DEFAULT is returned.

1.3.1.2.21.8 IG_load_color_reduction_set

This function instructs ImageGear to perform color reduction to reduce the number of Bits Per Pixel whenever loading an image whose bit depth is greater than that specified by nColorReduceMode.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_color_reduction_set(
   UINT nColorReduceMode
);
```

Arguments:

Name	Type	Description
nColorReduceMode	UINT	One of enumLoadColor enumeration values. IG_LOAD_COLOR_DEFAULT means that no color reduction is wanted.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Reduce 24-bit images to 8-bit when loading: */
AT_ERRCOUNT nErrCount = IG_load_color_reduction_set ( IG_LOAD_COLOR_8 );
```

Remarks:

The bit depth is reduced to 8, 4, or 1 as specified. Call with nColorReduceMode = IG_LOAD_COLOR_DEFAULT to disable color reduction.

See also IG_load_color_reduction_get() function.

1.3.1.2.21.9 IG_load_extra_mode_get

This function retrieves the last setting made by calling function IG_load_extra_mode_set().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_extra_mode_get(
   enumIGExtraMode* lpMode
);
```

Arguments:

Name	Type	Description
lpMode	enumIGExtraMode*	Pointer to an enumIGExtraMode variable to receive the current Extra channel loading mode setting.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumIGExtraMode ExtraMode;
AT_ERRCOUNT nErrCount = IG_load_extra_mode_get ( &ExtraMode );
```

1.3.1.2.21.10 IG_load_extra_mode_set

This function instructs ImageGear to load or ignore extra channels when loading an image that contains any extra channels.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_extra_mode_set(
   enumIGExtraMode Mode
);
```

Arguments:

Name	Type	Description
Mode	enumIGExtraMode	IG_EXTRA_MODE_KEEP (default) to load Extra channel if it is present; IG_EXTRA_MODE_IGNORE to ignore Extra channel. See enumIGExtraMode.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount = IG_load_extra_mode_set(IG_EXTRA_MODE_KEEP);
```

Remarks:

See also IG_load_extra_mode_get() function.

1.3.1.2.21.11 IG_load_FD

This function loads an image from a file into memory and creates a HIGEAR handle for the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_FD(
   AT_INT fd,
   LONG lOffset,
   UINT nPage,
   UINT nTile,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file containing the image to be loaded. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. lOffset is usually 0.
nPage	UINT	Page number to load if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
nTile	UINT	If loading an image that is tiled, you can set the number of a specific tile to load. Tile numbers begin at 1, not 0. Set to 1 for a non-tiled image.
lphIGear	LPHIGEAR	Pointer to HIGEAR object to which this function will return the HIGEAR handle of the image loaded.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR    hIGear = 0;     /* handle ret'd by IG_load_FD   */
HANDLE fd;                  /* File Descriptor */
LONG     lOffset;         /* offset to image in file   */
UINT     nPageNum;        /* will be 0 for this call  */
AT_ERRCOUNT nErrcount;    /* to test for errors  */

fd = CreateFile(_T("picture.bmp"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

if(fd != INVALID_HANDLE_VALUE)
{
    nPageNum  = 1;           /* not a multi-page file  */
    lOffset    = 0;          /* access file from start  */
    /* Load image, and obtain its HIGEAR handle:    */
    nErrcount = IG_load_FD((AT_INT)fd, lOffset, nPageNum, 0, &hIGear );
    CloseHandle(fd);
}
//...
```

```
// Destroy the image
if(IG_image_is_valid(hIGear))
{
    IG_image_delete(hIGear);
}
```

Remarks:

Unlike IG_load_file, this function is used when the file is already open and you have its File handle (fd). The HIGEAR handle, which ImageGear assigns for the loaded image, is returned to you via argument lphIGear. The file indicated by fd may be in any format recognized by ImageGear. IG_load_FD() will determine the format by inspecting the file's header section. See ImageGear Supported File Formats Reference.

Simply loading the file does not cause it to be displayed. Refer to IG_dspl_image_draw and related routines for information about how to display an image once it is in memory. See also IG_load_file_display.

lOffset represents the number of bytes, positive or negative, from the position in the file currently pointed to by fd. The fd may have been moved around a few times so that it is no longer pointing to the beginning of the file. Be sure to keep this in mind as you set the value of lOffset.

The nPage argument is set to 1 or greater if you are loading from a multi-page file to indicate which page (image) you want to load. Set nPage to 1 for a non-multi-page file.

If you set nPage to < 1, ImageGear will default the value to 1; if you set nPage to greater than the number of pages in the document, ImageGear will default the value to the last page number. This same default procedure applies to the nTile parameter as well.

If you wish to make a subsequent call to IG_info_get_FD_ex, you must first move the file pointer (of loaded image) to the beginning of the file, or you will receive an error. This happens because after an image is loaded, the file pointer is positioned at the end of the image in the file. To avoid the error:

- Call IG_load_FD().
- Call the appropriate C or Windows function that will set the pointer back to the beginning of the image's header information.
- Call IG_info_get_FD_ex().

See Also

IG_fltr_load_FD_format

1.3.1.2.21.12 IG_load_FD_CB

This function loads an image from a file using user-defined callback functions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_FD_CB(
   AT_INT fd,
   LONG lOffset,
   UINT nPage,
   UINT nTile,
   LPFNIG_RASTER_SET lpfnRasterSet,
   LPFNIG_DIB_CREATE lpfnDIBCreate,
   LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file containing the image to be loaded. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. lOffset is usually 0.
nPage	UINT	Page number to load if this is a multi-page (multi-image) file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
nTile	UINT	If loading an image that is tiled, you can set the number of a specific tile to load. Tile numbers begin at 1, not 0. Set to 1 for a non-tiled image.
lpfnRasterSet	LPFNIG_RASTER_SET	Pointer to callback function to be called after each raster line is read.
lpfnDIBCreate	LPFNIG_DIB_CREATE	Pointer to callback function to be called after the file header has been read.
lpPrivateData	LPVOID	Pointer to a private data area. This pointer will be passed to the callback functions.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

Actual set of pixel formats supported by this function can be narrower, depending on the implementation of the user-defined callback functions.

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_load_FD_CB_ex instead.

It is the responsibility of your two callback functions, lpfnDIBCreate and lpfnRasterSet, to create the DIB or other structure you want. Your lpfnDIBCreate callback function is called after the file's header has been read. Then your lpfnRasterSet callback function is called for each raster line read. See the descriptions under function types LPFNIG_DIB_CREATE and LPFNIG_RASTER_SET for how these callback functions are called.

If you want a HIGEAR handle for the DIB your callback functions have created, you can obtain one (after the load is complete) by calling function IG_image_DIB_import.

1.3.1.2.21.13 IG_load_FD_CB_ex

This function loads an image from a file using user-defined callback functions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_FD_CB_ex(
   AT_INT fd,
   LONG lOffset,
   UINT nPage,
   UINT nTile,
   LPFNIG_RASTER_SET lpfnRasterSet,
   LPFNIG_DIB_CREATE_EX lpfnDIBCreateEx,
   LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file containing the image to be loaded. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. lOffset is usually 0.
nPage	UINT	Page number to load if this is a multi-page (multi-image) file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
nTile	UINT	If loading an image that is tiled, you can set the number of a specific tile to load. Tile numbers begin at 1, not 0. Set to 1 for a non-tiled image.
lpfnRasterSet	LPFNIG_RASTER_SET	Pointer to callback function to be called after each raster line is read.
lpfnDIBCreateEx	LPFNIG_DIB_CREATE_EX	Pointer to callback function to be called after the file header has been read.
lpPrivateData	LPVOID	Pointer to a private data area. This pointer will be passed to the callback functions.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Actual set of pixel formats supported by this function can be narrower, depending on the implementation of the user-defined callback functions.

Example:

```
AT_ERRCOUNT ACCUAPI MyDIBCreateEx(
    LPVOID              lpPrivate, /* Private data passed in  */
    const HIGDIBINFO    hDIB       /* DIB info object for DIB */
    )
```

```
{
    /* Get info about image and allocate storage here */
    return 0;
}
AT_ERRCOUNT ACCUAPI MyRasterSet(
    LPVOID           lpPrivate,    /* Private data passed in  */
    const LPAT_PIXEL lpRaster,     /* Raster line to set      */
    AT_PIXPOS        row,          /* Y position in the image */
    DWORD            rasterSize    /* Size of the raster line */
    )
{
    /* Do something with incoming raster data here */
    return 0;
}

void Example_IG_load_FD_CB_ex()
{
    AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
    HANDLE fd;               /* File descriptor */
    DWORD dwPrivate[10];    /* Some private data */
    fd = CreateFile(_T("picture.bmp"), GENERIC_READ,
            0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

    if (fd != INVALID_HANDLE_VALUE)
    {
        nErrcount = IG_load_FD_CB_ex(
            (AT_INT)fd,       /* File descriptor             */
            0L,               /* Offset to image             */
            1,                /* Page number to load         */
            1,                /* Reserved. Always set to 1   */
            MyRasterSet,      /* Called for each raster line */
            MyDIBCreateEx,    /* Called after header is read */
            dwPrivate);       /* Callback data               */
        CloseHandle(fd);
    }
}
```

Remarks:

It is the responsibility of your two callback functions, lpfnDIBCreateEx and lpfnRasterSet, to create the image storage you want. Your lpfnDIBCreateEx callback function is called after the file's header has been read. Then your lpfnRasterSet callback function is called for each raster line read. See the descriptions under function types LPFNIG_DIB_CREATE_EX and LPFNIG_RASTER_SET for how these callback functions are called.

If you want a HIGEAR handle for the DIB your callback functions have created, you can obtain one (after the load is complete) by calling function IG_image_DIB_import.

1.3.1.2.21.14 IG_load_file

This function loads an image from the specified file into memory and creates a HIGEAR handle for the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_file(
   const LPSTR lpszFileName,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Name of image file (you may include path with filename) to load into memory.
lphIGear	LPHIGEAR	Pointer to HIGEAR object in which to return the ImageGear handle of the image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear = 0;             /* Will hold handle returned by IG_load_file*/
AT_ERRCOUNT nErrcount;     /* Count of errs on stack upon ret from func*/
/* Load image file "picture.bmp" from working directory, creating DIB */
/* and obtaining the image's ImageGear handle: */
nErrcount = IG_load_file ( "picture.bmp" , &hIGear );

//...

// Destroy the image
if(IG_image_is_valid(hIGear))
{
    IG_image_delete(hIGear);
}
```

Remarks:

The handle which ImageGear assigns for this image is returned to you in argument lphIGear. The file named by filename may be in any format recognized by ImageGear. IG_load_file() will determine the format by inspecting the file's header section. See ImageGear Supported File Formats Reference for information on image file formats supported by ImageGear.

Note that simply loading the file does not cause it to be displayed. Refer to IG_dspl_image_draw and related routines, for how to display an image once it is in memory. See also IG_load_file_display.

Some file formats, such as TXT, JPEG, and others, may be loaded with additional control, using IG_fltr_ctrl_get and IG_fltr_ctrl_set. See the description of these functions also in Using Format Filters API for Filter Control.

If the file pointed to by lpszFileName has multiple pages (images), the function will load the first page. To load pages of multi-page images, use IG_fltr_load_file, IG_load_FD or IG_load_mem.

1.3.1.2.21.15 IG_load_file_display

This function loads an image from the specified file into memory, creates HIGEAR handle for the image and simultaneously displays it.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_file_display(
   const LPSTR lpszFileName,
   DWORD dwGrpID,
   HWND hWnd,
   HDC hDC,
   LPFNIG_LOAD_DISP lpfnLoadDisp,
   LPVOID lpPrivateData,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Pointer to filename (including path if desired) of file to load and display.
dwGrpID	DWORD	Display group identifier that should be used for display operations.
hWnd	HWND	Handle of window where to draw image.
hDC	HDC	Windows Device Context of device or window in which to display image.
lpfnLoadDisp	LPFNIG_LOAD_DISP	Pointer to a callback function (or name of callback function) to call when image has been loaded, but before it is displayed.
lpPrivateData	LPVOID	Pointer to a private data area. This pointer will be passed to the callback function when it is called.
lphIGear	LPHIGEAR	pointer to a variable of type HIGEAR to hold the returned ImageGear HIGEAR handle of the newly loaded image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

Since you may want to set display attributes prior to displaying, ImageGear first loads the image's header, creating its HIGEAR handle, then calls your callback function (with the image's HIGEAR handle and your lpPrivate pointer) so you can set display attributes, device rectangle, image rectangle, or perform other operations. When your callback function returns, ImageGear then displays the image, one raster line at a time, as the image is loaded.

See the description for callback type LPFNIG_LOAD_DISP, and see also the section Displaying Images for a discussion of display attributes and how to set them.

1.3.1.2.21.16 IG_load_mem

This function loads an image from memory and creates HIGEAR handle for the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_mem(
   LPVOID lpImage,
   AT_UINT nSize,
   UINT nPage,
   UINT nTile,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to memory location of an image file that is currently in memory.
nSize	AT_UINT	Size of image in memory.
nPage	UINT	Page number to load if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
nTile	UINT	If loading an image that is tiled, you can set the number of a specific tile to load. Set to 1 for a non-tiled image.
lphIGear	LPHIGEAR	Pointer to HIGEAR object in which to return ImageGear handle of the newly loaded image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear = 0;       /* handle ret'd by ImageGear  */
char* lpWhereFile; /* ptr to image file in mem   */
AT_UINT nWholeSize;    /* Size of image in memory */
AT_ERRCOUNT nErrcount;  /* to test for errors         */

// Open a file and get its size
FILE* fd = NULL;
fopen_s(&fd, "picture.bmp", "rb");
if(fd != NULL)
{
    fseek(fd, 0, SEEK_END);
    nWholeSize = (AT_UINT)ftell(fd);
    fseek(fd, 0, SEEK_SET);
    // Allocate memory and read the image into the memory buffer
    lpWhereFile = (char*)malloc(nWholeSize);
    fread(lpWhereFile, 1, nWholeSize, fd);
    // File is no longer needed - close it
    fclose(fd);
    // Load image from the memory
    nErrcount = IG_load_mem(lpWhereFile, nWholeSize, 1, 0, &hIGear);
    // delete memory
    free(lpWhereFile);
}
```

```
//...

// Destroy the image
if(IG_image_is_valid(hIGear))
{
    IG_image_delete(hIGear);
}
```

Remarks:

The entire image file (even if a multi-page file) including header, is in memory. The format must be one of the file formats recognized by ImageGear. See ImageGear Supported File Formats Reference.

Argument lpImage is a pointer to the start of the image file in memory. dwSize is the size of the entire file (even if a multi-page file). For a multi-page file, nPage is the page number to load. Note that page numbers in multi-page files begin at 1, not 0. Set nPage = 1 if the file is a non-multi-page file.

This function creates a DIB for the image and loads the image into it. The handle which ImageGear assigns for this image is returned to you via argument lphIGear.

Note that simply loading the file does not cause it to be displayed. Refer to function IG_dspl_image_draw for how to display an image once it is in memory.

If you set nPage to < 1, ImageGear will default the value to 1; if you set nPage to greater than the number of pages in the document, ImageGear will default the value to the last page number. This same default procedure applies to the nTile parameter as well.

This function is very similar in operation to IG_load_file, except that the file to load from is located in memory rather than on a mass storage device.

1.3.1.2.21.17 IG_load_mem_CB

This function loads an image from a memory buffer using user-defined callback functions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_mem_CB(
   LPVOID lpImage,
   AT_UINT nSize,
   UINT nPage,
   UINT nTile,
   LPFNIG_RASTER_SET lpfnRasterSet,
   LPFNIG_DIB_CREATE lpfnDIBCreate,
   LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to a memory buffer containing the image.
nSize	AT_UINT	Size of image in memory.
nPage	UINT	Page number to load if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
nTile	UINT	If loading an image that is tiled, you can set the number of a specific tile to load. Set to 1 for a non-tiled image.
lpfnRasterSet	LPFNIG_RASTER_SET	Pointer to callback function to be called after each raster line is read.
lpfnDIBCreate	LPFNIG_DIB_CREATE	Pointer to callback function to be called after the file header has been read.
lpPrivateData	LPVOID	Pointer to a private data area. This pointer will be passed to the callback functions.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

Actual set of pixel formats supported by this function can be narrower, depending on the implementation of the user-defined callback functions.

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_load_mem_CB_ex instead.

See the description under function IG_load_FD_CB. See also function IG_load_mem.

If you set nPage to < 1, ImageGear will default the value to 1; if you set nPage to greater than the number of pages in the document, ImageGear will default the value to the last page number. This same default procedure applies to the nTile parameter as well.

1.3.1.2.21.18 IG_load_mem_CB_ex

This function loads an image from a file using user-defined callback functions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_mem_CB_ex(
   LPVOID lpImage,
   AT_UINT nSize,
   UINT nPage,
   UINT nTile,
   LPFNIG_RASTER_SET lpfnRasterSet,
   LPFNIG_DIB_CREATE_EX lpfnDIBCreateEx,
   LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to a memory buffer containing the image.
nSize	AT_UINT	Size of image in memory.
nPage	UINT	Page number to load if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
nTile	UINT	If loading an image that is tiled, you can set the number of a specific tile to load. Tile numbers begin at 1, not 0. Set to 1 for a non-tiled image.
lpfnRasterSet	LPFNIG_RASTER_SET	Pointer to callback function to be called after each raster line is read.
lpfnDIBCreateEx	LPFNIG_DIB_CREATE_EX	Pointer to callback function to be called after the file header has been read.
lpPrivateData	LPVOID	Pointer to a private data area. This pointer will be passed to the callback functions.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Actual set of pixel formats supported by this function can be narrower, depending on the implementation of the user-defined callback function.

Example:

```
AT_ERRCOUNT ACCUAPI MyDIBCreateEx(
    LPVOID             lpPrivate, /* Private data passed in  */
    const HIGDIBINFO   hDIB       /* DIB info object for DIB */
    )
{
    /* Get info about image and allocate storage here */
    return 0;
}
AT_ERRCOUNT ACCUAPI MyRasterSet(
    LPVOID           lpPrivate,   /* Private data passed in  */
    const LPAT_PIXEL lpRaster,    /* Raster line to set      */
    AT_PIXPOS        row,          /* Y position in the image */
    DWORD            rasterSize    /* Size of the raster line */
    )
{
    /* Do something with incoming raster data here */
    return 0;
}

void Example_IG_load_FD_CB_ex()
{
    AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
    HANDLE fd;               /* File descriptor */
    DWORD dwPrivate[10];    /* Some private data */
    fd = CreateFile(_T("picture.bmp"), GENERIC_READ,
            0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

    if (fd != INVALID_HANDLE_VALUE)
    {
        nErrcount = IG_load_FD_CB_ex(
            (AT_INT)fd,        /* File descriptor              */
            0L,                /* Offset to image              */
            1,                 /* Page number to load          */
            1,                 /* Tile number to load          */
            MyRasterSet,       /* Called for each raster line */
            MyDIBCreateEx,     /* Called after header is read */
            dwPrivate);        /* Callback data                */
        CloseHandle(fd);
    }
}
```

If you set nPage to < 1, ImageGear will default the value to 1; if you set nPage to greater than the number of pages in the document, ImageGear will default the value to the last page number. This same default procedure applies to the nTileNum parameter as well.

See Also

IG_load_FD_CB

IG_load_mem

1.3.1.2.21.19 IG_load_raw_FD

This function creates an ImageGear image from the raw image data of the file whose File handle is fd.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_raw_FD(
   AT_INT fd,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight,
   UINT nBitsPerPixel,
   AT_MODE nFillOrder,
   AT_MODE nCompression,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file containing the image to be loaded. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
nWidth	AT_DIMENSION	The width, in pixels, of the image.
nHeight	AT_DIMENSION	The height, in pixels, of the image.
nBitsPerPixel	UINT	The Bits Per Pixel of the raw data to load.
nFillOrder	AT_MODE	Set to the fill order used in the image: Least Significant Bit first (LSB) or Most Significant Bit first (MSB). Use one of the ImageGear - defined constants: IG_FILL_LSB or IG_FILL_MSB. • For G3/G4 compressed data, this parameter specifies the Least/Most significant bit. • For 16 bit grayscale uncompressed images, this parameter specifies the Least/Most significant byte. • For any other bit depths and compressions, this parameter is ignored.
nCompression	AT_MODE	Compression used by the RAW image data. Set to one of enumIGCompressions constants.
lphIGear	LPHIGEAR	Returns a HIGEAR handle to the raw image data just loaded.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp
- Grayscale - 12, 16 bpp
- RGB - 24 bpp
- CMYK - 32 bpp

Example:

```
HIGEAR hIGear = 0;              // Will receive HIGEAR image handle
HANDLE fd;                      // File Descriptor handle

// Open a dile to read
```

```
fd = CreateFile(_T("Group4.raw"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

if(fd != INVALID_HANDLE_VALUE)
{
    AT_ERRCOUNT nErrcount;  // Returned count of errors
    nErrcount = IG_load_raw_FD((AT_INT)fd, 2320, 3408, 1, IG_FILL_MSB,
IG_COMPRESSION_CCITT_G4, &hIGear);

    //...

    // Destroy the image
    if(IG_image_is_valid(hIGear))
    {
        IG_image_delete(hIGear);
    }
    CloseHandle(fd);
}
```

Remarks:

A raw image file contains no header or identifying information. You must supply this function with all of the information needed to correctly parse the image data, including the compression, byte fill order, width, height, and bit depth. Currently, this function can be used to read raw image data from the following types of files: ABIC, CCITT - Group 3, Group 3 2D, Group 4, LZW, and raw uncompressed data.

The ABIC and LZW compression types are available as separate components to ImageGear. See ImageGear Components for details on working with ImageGear components.

The pointer must be positioned at the start of the data. (For example, your application should read or seek past any header that is present.)

Additionally, you can specify row and pixel alignment for the loading of uncompressed images using ALIGNMENT and UNCOMPRESSED_PACKED image control parameters, respectively. See RAW format reference for more information.

For uncompressed images only, ImageGear's Load Raw functions consider the coordinates (0,0) to refer to the lower-left corner of the bitmap.

1.3.1.2.21.20 IG_load_raw_file

This function loads a raw (no header) image data file from disk.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_raw_file(
   const LPSTR lpszFileName,
   LONG lOffset,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight,
   UINT nBitsPerPixel,
   AT_MODE nFillOrder,
   AT_MODE nCompression,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Name of the file.
lOffset	LONG	Offset in the file from where the raw image data starts.
nWidth	AT_DIMENSION	The width, in pixels, of the image.
nHeight	AT_DIMENSION	The height, in pixels, of the image.
nBitsPerPixel	UINT	The Bits Per Pixel of the raw data to load.
nFillOrder	AT_MODE	Set to the fill order used in the image: Least Significant Bit first (LSB) or Most Significant Bit first (MSB). Use one of the ImageGear - defined constants: IG_FILL_LSB or IG_FILL_MSB. • For G3/G4 compressed data, this parameter specifies the Least/Most significant bit. • For 16 bit grayscale uncompressed images, this parameter specifies the Least/Most significant byte. • For any other bit depths and compressions, this parameter is ignored.
nCompression	AT_MODE	Compression used by the RAW image data. Set to one of enumIGCompressions constants.
lphIGear	LPHIGEAR	Returns a HIGEAR handle to the raw image data just loaded.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp
- Grayscale - 12, 16 bpp
- RGB - 24 bpp
- CMYK - 32 bpp

Example:

```
HIGEAR hIGear = 0;                  // Will receive HIGEAR image handle

AT_ERRCOUNT nErrcount;  /* Returned count of errors */
nErrcount = IG_load_raw_file("Group4.raw", 0, 2320, 3408, 1, IG_FILL_MSB,
IG_COMPRESSION_CCITT_G4, &hIGear );
```

```
//...

// Destroy the image
if(IG_image_is_valid(hIGear))
{
    IG_image_delete(hIGear);
}
```

Remarks:

A raw image file contains no header or identifying information. You must supply this function with all of the information needed to correctly parse the image data, including the offset to the start of the pixel data, compression, byte fill order, width, height, and bit depth. Currently, this function can be used to read raw image data from the following types of files: ABIC, CCITT - Group 3, Group 3 2D, Group 4, LZW, and raw uncompressed data.

The ABIC and LZW compression types are available as separate components to ImageGear. See ImageGear Components for details on working with ImageGear components.

Additionally, you can specify row and pixel alignment for the loading of uncompressed images using ALIGNMENT and UNCOMPRESSED_PACKED image control parameters, respectively. See RAW format reference for more information.

For uncompressed images only, ImageGear's Load Raw functions consider the coordinates (0,0) to refer to the lower-left corner of the bitmap.

1.3.1.2.21.21 IG_load_raw_mem

This function loads an image from the raw image data located in the memory buffer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_raw_mem(
   LPVOID lpImage,
   AT_UINT nSize,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight,
   UINT nBitsPerPixel,
   AT_MODE nFillOrder,
   AT_MODE nCompression,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to memory area from which to load.
nSize	AT_UINT	Size of image in memory.
nWidth	AT_DIMENSION	The width, in pixels, of the image.
nHeight	AT_DIMENSION	The height, in pixels, of the image.
nBitsPerPixel	UINT	The Bits Per Pixel of the raw data to load.
nFillOrder	AT_MODE	Set to the fill order used in the image: Least Significant Bit first (LSB) or Most Significant Bit first (MSB). Use one of the ImageGear - defined constants: IG_FILL_LSB or IG_FILL_MSB. • For G3/G4 compressed data, this parameter specifies the Least/Most significant bit. • For 16 bit grayscale uncompressed images, this parameter specifies the Least/Most significant byte. • For any other bit depths and compressions, this parameter is ignored.
nCompression	AT_MODE	Compression used by the RAW image data. Set to one of enumIGCompressions constants.
lphIGear	LPHIGEAR	Returns you a HIGEAR handle to the raw image data just loaded.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp
- Grayscale - 12, 16 bpp
- RGB - 24 bpp
- CMYK - 32 bpp

Example:

```
HIGEAR hIGear = 0;                // Will receive HIGEAR image handle
char* buffer = "ï¿½AAAAAAA";        // Buffer with image data
AT_DIMENSION nWidth = 2;        // Pixels per row of data to read
AT_DIMENSION nHeight = 2;        // Rows to read
AT_MODE nCompression = IG_COMPRESSION_NONE; // Compression
AT_MODE nFillOrder = IG_FILL_MSB;            // Fill order
```

```
UINT nBitsPerPixel = 16;         // Bit depth

AT_ERRCOUNT nErrcount;         /* Returned count of errors */
nErrcount = IG_load_raw_mem (buffer, (AT_UINT)strlen(buffer), nWidth, nHeight,
nBitsPerPixel, nFillOrder, nCompression,
    &hIGear );

//...

// Destroy the image
if(IG_image_is_valid(hIGear))
{
    IG_image_delete(hIGear);
}
```

Remarks:

A raw image file contains no header or identifying information. You must supply this function with all of the information needed to correctly parse the image data, including the compression, byte fill order, width, height, and bit depth. Currently, this function can be used to read raw image data from the following types of files: ABIC, CCITT - Group 3, Group 3 2D, Group 4, LZW, and raw uncompressed data.

The ABIC and LZW compression types are available as separate components to ImageGear. See ImageGear Components for details on working with ImageGear components.

Note that lpImage must point to the start of the actual data (not to the start of any header information that may be present).

Additionally, you can specify row and pixel alignment for the loading of uncompressed images using ALIGNMENT and UNCOMPRESSED_PACKED image control parameters, respectively. See RAW format reference for more information.

For uncompressed images only, ImageGear's Load Raw functions consider the coordinates (0,0) to refer to the lower- left corner of the bitmap.

1.3.1.2.21.22 IG_load_rect_get

This function obtains the current load rectangle, as set in the last call to IG_load_rect_set.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_rect_get(
   LPAT_PIXPOS lpX,
   LPAT_PIXPOS lpY,
   LPAT_DIMENSION lpWidth,
   LPAT_DIMENSION lpHeight
);
```

Arguments:

Name	Type	Description
lpX	LPAT_PIXPOS	Pointer to variable of type AT_PIXPOS to receive X coordinate of load rectangle.
lpY	LPAT_PIXPOS	Pointer to AT_PIXPOS variable to receive Y coordinate of load rectangle.
lpWidth	LPAT_DIMENSION	Pointer to variable of type AT_DIMENSION to receive width of load rectangle.
lpHeight	LPAT_DIMENSION	Pointer to variable of type AT_DIMENSION to receive height of load rectangle.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_PIXPOS nXLoadCrop,        /* X coordinate of load crop rectangle  */
nYLoadCrop;                    /* Y coordinate  */
AT_DIMENSION
    nWidCrop,                /* Width of load crop rectangle  */
    nHiCrop;                /* Height   */
AT_ERRCOUNT nErrcount;        /* Returned count of errors */
nErrcount = IG_load_rect_get ( &nXLoadCrop, &nYLoadCrop, &nWidCrop, &nHiCrop );
```

1.3.1.2.21.23 IG_load_rect_set

This function sets the load rectangle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_rect_set(
   AT_PIXPOS nX,
   AT_PIXPOS nY,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight
);
```

Arguments:

Name	Type	Description
nX	AT_PIXPOS	X coordinate of load rectangle.
nY	AT_PIXPOS	Y coordinate of load rectangle.
nWidth	AT_DIMENSION	Width of load rectangle.
nHeight	AT_DIMENSION	Height of load rectangle.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Set to only load the upper left 1000 x 500 of the image: */
IG_load_rect_set ( 0, 0, 1000, 500 );
```

Remarks:

This function will cause an image loaded to be cropped. Portions of the image falling outside the coordinates of this rectangle will be discarded, and will not appear in the image bitmap of the DIB created by the load.

To reset ImageGear to its default behavior of loading the whole image, set all parameters to 0. See also IG_load_rect_get function.

1.3.1.2.21.24 IG_load_size_get

This function obtains the load size dimensions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_size_get(
   LPAT_DIMENSION lpWidth,
   LPAT_DIMENSION lpHeight
);
```

Arguments:

Name	Type	Description
lpWidth	LPAT_DIMENSION	Pointer to variable of type AT_DIMENSION to receive the load size width.
lpHeight	LPAT_DIMENSION	Pointer to variable of type AT_DIMENSION to receive the load size height.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_DIMENSION nWid, nHi;   /* Will receive current load size settings */
AT_ERRCOUNT nErrCount = IG_load_size_get ( &nWid, &nHi );
```

Remarks:

To reset ImageGear so that it will default to normal resolution, set both parameters to 0 using IG_load_size_set.

1.3.1.2.21.25 IG_load_size_set

This function instructs ImageGear to resize images to specified dimensions during the loading.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_size_set(
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight
);
```

Arguments:

Name	Type	Description
nWidth	AT_DIMENSION	Specifies the width the images should have after loading.
nHeight	AT_DIMENSION	Specifies the height the images should have after loading.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* On loading, map the image into a 512 x 256 image bitmap: */
IG_load_size_set ( 512, 256 );
```

Remarks:

ImageGear will resize the images according to the specified dimensions, regardless of the dimensions in the source image file. The effect is similar to loading and then resizing the image. To reset ImageGear so that it will default to loading the entire image, set both parameters to 0. See also IG_load_size_get.

ImageGear does not apply interpolation for image resizing during the loading.

1.3.1.2.21.26 IG_load_tag_CB_register

This function has been deprecated and will be removed from the public API in a future release. Please use IG_fltr_metad_callback_get instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_tag_CB_register(
   LPFNIG_TAG_SET lpfnTagSet,
   LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
lpfnTagSet	LPFNIG_TAG_SET	Pointer to callback function to be called with each Tag encountered while loading.
lpPrivateData	LPVOID	Pointer to private data (passed to callback function).

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
IG_load_tag_CB_register(NULL, NULL);
```

Remarks:

This function registers a callback function of type LPFNIG_TAG_GET, LPFNIG_TAG_SET, or LPFNIG_TAG_USER_GET.

See the Core Component Callback Functions Reference section. These callback function types are defined by ImageGear to take a certain set of parameters and to return data to you. Your callback function must supply ImageGear with the type of data required by the callback that you choose.

Once you have written a callback function in one of the types listed above, this function should be called to register it. Once registered, your function will be called once for each Tag encountered while loading a file. A "tag" may also be known as an element in the image's header, or as non-image data. The TIFF format popularized the use of the word "tag".

Different file formats have different sets of tags.

1.3.1.2.21.27 IG_load_thumbnail

This function loads a thumbnail (if one exists in the file) from file lpszFileName, and returns the HIGEAR handle of the resulting image to the HIGEAR object pointed to by argument lphIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_thumbnail(
   const LPSTR szFileName,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
szFileName	const LPSTR	Name of file.
lphIGear	LPHIGEAR	Pointer to HIGEAR variable to receive handle.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

1.3.1.2.21.28 IG_load_thumbnail_FD

This function loads a thumbnail from the file specified by a file handle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_thumbnail_FD(
   AT_INT fd,
   LONG lOffset,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file containing the image from which to load the thumbnail. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset in file at which file image begins.
lphIGear	LPHIGEAR	Pointer to a HIGEAR variable to receive the handle.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear1 = 0;      /* handle ret'd by IG_load_FD  */
HIGEAR hIGear2 = 0;      /* handle for thumbnail  */
HANDLE fd;                 /* DOS File Descriptor  */
LONG     lOffset;            /* offset to image in file */
UINT     nPageNum;           /* will be 0 for this call */
AT_ERRCOUNT nErrcount;         /* to test for errors  */

fd = CreateFile(_T("picture.bmp"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

if(fd != INVALID_HANDLE_VALUE)
{
    nPageNum  = 0;           /* not a multi-page file  */
    lOffset    = 0;          /* access file from start */
    /* Load image, and obtain its HIGEAR handle:    */
    nErrcount = IG_load_FD((AT_INT)fd, lOffset, nPageNum, 0, &hIGear1 );
    CloseHandle(fd);
    if(nErrcount == 0)
    {
        nErrcount = IG_load_thumbnail_FD((AT_INT)fd, lOffset, &hIGear2 );
    }

    //...

    // Destroy images
    if(IG_image_is_valid(hIGear1))
    {
        IG_image_delete(hIGear1);
    }
    if(IG_image_is_valid(hIGear2))
    {
        IG_image_delete(hIGear2);
    }
}
```

Remarks:

The handle of the resulting image is returned to the HIGEAR variable pointed to by argument lphIGear. See also functions IG_load_thumbnail, and IG_save_thumbnail_set.

1.3.1.2.21.29 IG_load_thumbnail_mem

This function loads a thumbnail from the image located in a memory buffer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_thumbnail_mem(
   LPVOID lpImage,
   AT_UINT nSize,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to start of the image in memory.
nSize	AT_UINT	Size of image in memory.
lphIGear	LPHIGEAR	Pointer to HIGEAR variable to receive handle.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear1 = 0;        /* handle ret'd by IG_load_mem  */
HIGEAR hIGear2 = 0;        /* ret'd by IG_load_thumbnail_mem  */
char far * lpWhereFile = NULL;     /* ptr to image file in mem   */
AT_UINT  nImageSize;   /* Size of image in memory */
AT_ERRCOUNT nErrcount;/* to test for errors          */

// Open a file and get its size
FILE* fd;
fopen_s(&fd, "picture.bmp", "rb");
if(fd != NULL)
{
    fseek(fd, 0, SEEK_END);
    nImageSize = (AT_UINT)ftell(fd);
    fseek(fd, 0, SEEK_SET);
    // Allocate memory and read the image into the memory buffer
    lpWhereFile = (char*)malloc(nImageSize);
    fread(lpWhereFile, 1, nImageSize, fd);
    // File is no longer needed - close it
    fclose(fd);
}
// Load image from the memory
nErrcount = IG_load_mem(lpWhereFile, nImageSize, 1, 0, &hIGear1);
if ( nErrcount == 0 )
{
    nErrcount = IG_load_thumbnail_mem(lpWhereFile, nImageSize, &hIGear2);
}
// delete memory
if(lpWhereFile)
{
    free(lpWhereFile);
}

//...

// Destroy images
if(IG_image_is_valid(hIGear1))
{
    IG_image_delete(hIGear1);
}
if(IG_image_is_valid(hIGear2))
{
    IG_image_delete(hIGear2);
}
```

Remarks:

The handle of the resulting image is returned to the HIGEAR variable pointed to by argument lphIGear. See also functions IG_load_thumbnail, and IG_save_thumbnail_set.

1.3.1.2.21.30 IG_load_tiles_stitch

This function loads and stitches together a tiled image, returning a HIGEAR handle to the image in memory.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch(
   const LPSTR lpszFileName,
   UINT nPage,
   LPAT_STITCH lpStitch,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Set to the filename of the image to load and stitch.
nPage	UINT	Page number to load and stitch if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
lpStitch	LPAT_STITCH	Set to a structure of type AT_STITCH, which defines the reference tile number, and the number of row and columns of tiles.
lphIGear	LPHIGEAR	ImageGear returns a HIGEAR handle to the newly stitched image in memory.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrCount;
HIGEAR  hIGear = 0;
AT_STITCH stitch = {1, 1, 1};
CHAR* szFile = "picture.tif";
nErrCount = IG_load_tiles_stitch(szFile, 1, &stitch, &hIGear);

//...

// Destroy the image
if(IG_image_is_valid(hIGear))
{
    IG_image_delete(hIGear);
}
```

Remarks:

The AT_STITCH structure allows you to supply ImageGear with information on which tiles to use as the upper-left corner in the new stitched image, and how many tile rows and columns should be stitched together. For a graphical representation of how this works, see Working with Tiled Images.

The nPage argument is set to 1 or greater if you are loading from a multi-page file, to indicate which page (image) you want to load. Set nPage to 1 for a non-multi-page file.

If you set nPage to < 1, ImageGear will default the value to 1; if you set nPage to greater than the number of pages in the document, ImageGear will default the value to the last page number.

See also IG_load_tiles_stitch_FD, IG_load_tiles_stitch_mem, IG_tile_count_get functions.

For a complete discussion of working with tiled images see Working with Tiled Images.

1.3.1.2.21.31 IG_load_tiles_stitch_FD

This function loads and stitches together a tiled image from the file specified by the file handle, returning you a HIGEAR handle to the image in memory.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch_FD(
   AT_INT fd,
   LONG lOffset,
   UINT nPage,
   LPAT_STITCH lpStitch,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file containing the image to be loaded. This handle can be obtained from Microsoft Windows functions such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
lOffset	LONG	Offset into the file, in bytes, to where the image begins. This is the offset to the beginning of the header, not to the beginning of the bitmap. lOffset is usually 0.
nPage	UINT	Page number to load if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
lpStitch	LPAT_STITCH	Set to a structure of type AT_STITCH, which defines the reference tile number, and the number of rows and columns of tiles.
lphIGear	LPHIGEAR	Returns a HIGEAR handle to the newly stitched-together image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR    hIGear = 0;      /* handle ret'd by IG_load_FD */
HANDLE      fd;          /* File Descriptor  */
AT_ERRCOUNT nErrcount;   /* to test for errors  */
AT_STITCH  stitchStruct = {1, 1, 1};

fd = CreateFile(_T("picture.tif"), GENERIC_READ,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
if(fd != INVALID_HANDLE_VALUE)
{
    /* Load tiles, stitch together and return HIGEAR handle:  */
    nErrcount = IG_load_tiles_stitch_FD((AT_INT)fd, 0, 1, &stitchStruct, &hIGear );
    CloseHandle(fd);

    //...

    // Destroy the image
    if(IG_image_is_valid(hIGear))
    {
        IG_image_delete(hIGear);
    }
}
```

Remarks:

Unlike IG_load_tiles_stitch, this function is used when the file is already open and you have its File handle.

The AT_STITCH structure allows you to tell ImageGear which tile to use as the upper-left corner in the new stitched image, and how many tile rows and columns should be stitched together. For a graphical representation of how this works, see Working with Tiled Images.

The nPage argument is set to 1 or greater if you are loading from a multi-page file, to indicate which page (image) you want to load. Set nPage to 1 for a non multi-page file.

If you set nPage to < 1, ImageGear will default the value to 1; if you set nPage to greater than the number of pages in the document, ImageGear will default the value to the last page number.

See also IG_load_tiles_stitch, IG_load_tiles_stitch_mem, and IG_tile_count_get_FD functions.

For a complete discussion of working with tiled images, see Working with Tiled Images.

1.3.1.2.21.32 IG_load_tiles_stitch_mem

This function loads and stitches together a tiled image that has already been loaded into memory, returning you a HIGEAR handle to the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_load_tiles_stitch_mem(
   LPVOID lpImage,
   AT_UINT nSize,
   UINT nPage,
   LPAT_STITCH lpStitch,
   LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Pointer to a memory buffer containing the image.
nSize	AT_UINT	Size of image in memory.
nPage	UINT	Page number to load if this is a multi-page file. Note that page numbers begin at 1, not 0. Set nPage to 1 if this is not a multi-page file.
lpStitch	LPAT_STITCH	Set to a structure of type AT_STITCH, which defines the reference tile number, and the number of rows and columns of tiles.
lphIGear	LPHIGEAR	Returns a HIGEAR handle to the newly stitched-together image.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT  nErrcount;
HIGEAR hIGear;
char far * lpWhereFile;
AT_UINT         nWholeSize;
AT_STITCH stitchStruct = {1, 1, 1};
// Open a file and get its size
FILE* fd;
fopen_s(&fd, "picture.tif", "rb");
if(fd != NULL)
{
    fseek(fd, 0, SEEK_END);
    nWholeSize = (AT_UINT)ftell(fd);
    fseek(fd, 0, SEEK_SET);
    // Allocate memory and read the image into the memory buffer
    lpWhereFile = (char*)malloc(nWholeSize);
    fread(lpWhereFile, 1, nWholeSize, fd);
    // File is no longer needed - close it
    fclose(fd);
}

if(lpWhereFile != NULL)
{
    nErrcount = IG_load_tiles_stitch_mem(lpWhereFile, nWholeSize, 1, &stitchStruct,
&hIGear);
    // delete memory
    free(lpWhereFile);
}

//...

// Destroy the image
if(IG_image_is_valid(hIGear))
{
    IG_image_delete(hIGear);
}
```

Remarks:

The AT_STITCH structure allows you to tell ImageGear which tile to use as the upper-left corner in the new stitched image, and how many tile rows and columns should be stitched together. For a graphical representation of how this works, see Working with Tiled Images.

Simply loading and stitching the file does not cause it to be displayed. Refer to IG_dspl_image_draw and related routines, for how to display an image once it is in memory. See also IG_load_file_display function.

The nPage argument is set to 1 or greater if you are loading from a multi-page file, to indicate which page (image) you want to load. Set nPage to 1 for a non multi-page file.

If you set nPage to < 1, ImageGear will default the value to 1; if you set nPage to greater than the number of pages in the document, ImageGear will default the value to the last page number.

See also IG_load_tiles_stitch, IG_load_tiles_stitch_FD, IG_tile_count_get_memfunctions.

For a complete discussion of working with tiled images, see Working with Tiled Images.

1.3.1.2.22 LUT Functions

This section provides information about the LUT group of functions.

- IG_LUT_copy
- IG_LUT_copy_to_byte_array
- IG_LUT_copy_to_word_array
- IG_LUT_create
- IG_LUT_destroy
- IG_LUT_input_depth_get
- IG_LUT_input_is_signed_get
- IG_LUT_is_valid
- IG_LUT_item_get
- IG_LUT_item_set
- IG_LUT_length_get
- IG_LUT_output_depth_get
- IG_LUT_output_is_signed_get
- IG_LUT_size_get
- IG_LUT_update_from_byte_array
- IG_LUT_update_from_word_array

1.3.1.2.22.1 IG_LUT_copy

This function copies the LUT to a new HIGLUT object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_copy(
        HIGLUT SrcLUT,
        HIGLUT* lpDstLUT
);
```

Arguments:

Name	Type	Description
SrcLUT	HIGLUT	Source LUT handle.
lpDstLUT	HIGLUT*	New LUT handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.22.2 IG_LUT_copy_to_byte_array

This function copies a LUT to a byte array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_copy_to_byte_array(
        HIGLUT SrcLUT,
        AT_INT ArrayLength,
        AT_BYTE* lpArray
);
```

Arguments:

Name	Type	Description
SrcLUT	HIGLUT	LUT to be copied.
ArrayLength	AT_INT	Array length.
lpArray	AT_BYTE*	Byte array.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Array length must be equal to 2^InputDepth, where InputDepth is the input depth of the LUT.

1.3.1.2.22.3 IG_LUT_copy_to_word_array

This function copies LUT a to word array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_copy_to_word_array(
        HIGLUT SrcLUT,
        AT_INT ArrayLength,
        AT_WORD* lpArray
);
```

Arguments:

Name	Type	Description
SrcLUT	HIGLUT	LUT to be copied.
ArrayLength	AT_INT	Array length.
lpArray	AT_WORD*	Word array.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Array length must be equal to 2^InputDepth, where InputDepth is the input depth of the LUT.

1.3.1.2.22.4 IG_LUT_create

This function creates a LUT object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_create(
        AT_INT InputDepth,
        AT_BOOL InputIsSigned,
        AT_INT OutputDepth,
        AT_BOOL OutputIsSigned,
        HIGLUT* lpLUT
);
```

Arguments:

Name	Type	Description
InputDepth	AT_INT	Input bit count of the LUT.
InputIsSigned	AT_BOOL	Shows whether input of the LUT is signed.
OutputDepth	AT_INT	Output bit count of the LUT.
OutputIsSigned	AT_BOOL	Shows whether output of the LUT is signed.
lpLUT	HIGLUT*	Handle of the created LUT object.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.22.5 IG_LUT_destroy

This function destroys a LUT object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_destroy(
        HIGLUT lut
);
```

Arguments:

Name	Type	Description
lut	HIGLUT	Handle of the LUT to be destroyed.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.22.6 IG_LUT_input_depth_get

This function returns input bit count of the LUT.

Declaration:

```
AT_INT ACCUAPI IG_LUT_input_depth_get(HIGLUT lut);
```

Arguments:

Name	Type	Description
lut	HIGLUT	LUT handle.

Return Value:

Input bit count of the LUT.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Input depth defines the length (number of entries) of the lut, as follows:

LUTLength = 2^InputDepth.

1.3.1.2.22.7 IG_LUT_input_is_signed_get

This function tells whether input of the LUT is signed.

Declaration:

```
AT_BOOL ACCUAPI IG_LUT_input_is_signed_get(HIGLUT lut);
```

Arguments:

Name	Type	Description
lut	HIGLUT	LUT handle.

Return Value:

TRUE if the input is signed; FALSE - otherwise.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.22.8 IG_LUT_is_valid

This function checks whether HIGLUT is valid.

Declaration:

```
AT_BOOL ACCUAPI IG_LUT_is_valid(HIGLUT lut);
```

Arguments:

Name	Type	Description
lut	HIGLUT	LUT handle.

Return Value:

TRUE if HIGLUT is valid; FALSE - otherwise.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.22.9 IG_LUT_item_get

This function returns LUT item.

Declaration:

```
AT_INT ACCUAPI IG_LUT_item_get(
        HIGLUT lut,
        AT_INT32 index
);
```

Arguments:

Name	Type	Description
lut	HIGLUT	Handle of the LUT.
index	AT_INT32	Index of the item to be returned.

Return Value:

LUT item.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If LUT input is unsigned, valid values for index are in range [0, 2^InputDepth-1].

If LUT input is signed, valid values for index are in range [-2^(InputDepth-1), 2^(InputDepth-1)-1].

1.3.1.2.22.10 IG_LUT_item_set

This function sets the LUT item.

Declaration:

```
AT_VOID ACCUAPI IG_LUT_item_set(
        HIGLUT lut,
        AT_INT32 index,
        AT_INT value
);
```

Arguments:

Name	Type	Description
lut	HIGLUT	LUT handle.
index	AT_INT32	Index of the item.
value	AT_INT	Value of the item.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If LUT input is unsigned, valid values for index are in range [0, 2^InputDepth-1].

If LUT input is signed, valid values for index are in range [-2^(InputDepth-1), 2^(InputDepth-1)-1].

1.3.1.2.22.11 IG_LUT_length_get

This function returns the number of entries in the LUT.

Declaration:

```
AT_INT ACCUAPI IG_LUT_length_get(HIGLUT hlut);
```

Arguments:

Name	Type	Description
hlut	HIGLUT	LUT handle.

Return Value:

Returns the number of entries in the LUT.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.22.12 IG_LUT_output_depth_get

This function returns output bit count of the LUT.

Declaration:

```
AT_INT ACCUAPI IG_LUT_output_depth_get(HIGLUT lut);
```

Arguments:

Name	Type	Description
lut	HIGLUT	LUT handle.

Return Value:

Returns output bit count of the LUT.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Output depth defines the size of each entry of the LUT.

1.3.1.2.22.13 IG_LUT_output_is_signed_get

This function tells whether output of the LUT is signed.

Declaration:

```
AT_BOOL ACCUAPI IG_LUT_output_is_signed_get(HIGLUT lut);
```

Arguments:

Name	Type	Description
lut	HIGLUT	LUT handle.

Return Value:

TRUE - output of the LUT is signed; FALSE - otherwise.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.22.14 IG_LUT_size_get

This function returns the size, in bytes, of the LUT data.

Declaration:

```
AT_INT ACCUAPI IG_LUT_size_get(HIGLUT hlut);
```

Arguments:

Name	Type	Description
hlut	HIGLUT	Handle of the LUT.

Return Value:

Returns the size, in bytes, of the LUT data.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.22.15 IG_LUT_update_from_byte_array

This function updates LUT from a byte array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_update_from_byte_array(
        HIGLUT lut,
        AT_INT ArrayLength,
        const AT_BYTE* lpArray
);
```

Arguments:

Name	Type	Description
lut	HIGLUT	LUT handle.
ArrayLength	AT_INT	Array length.
lpArray	const AT_BYTE*	Byte array.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Array length must be equal to 2^InputDepth, where InputDepth is the input depth of the lut.

1.3.1.2.22.16 IG_LUT_update_from_word_array

This function updates LUT from a word array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_LUT_update_from_word_array(
        HIGLUT lut,
        AT_INT ArrayLength
        const AT_WORD* lpArray
);
```

Arguments:

Name	Type	Description
lut	HIGLUT	LUT handle.
ArrayLength	AT_INT	Number of WORDs in the array.
lpArray	const AT_WORD*	Word array.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Array length must be equal to 2^InputDepth, where InputDepth is the input depth of the LUT.

1.3.1.2.23 Mac Initialize and Close Functions

This section provides information about the Mac Initialize and Close group of functions.

- IG_initialize
- IG_close

1.3.1.2.23.1 IG_initialize

This function must be called before any other ImageGear function is used. This function initializes the ImageGear library. The lpData parameter is not used in current version and must be set to NULL.

Declaration:

```
AT_ERRCODE IG_initialize (LPVOID lpData);
```

Arguments:

Name	Type	Description
lpData	LPVOID	This argument currently isn't used, but is reserved for future use. Must be set to NULL for now.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Initialize ImageGear */
nErrCount = IG_initialize(NULL);
```

Sample:

ImageGearDemo

See Also:

IG_close()

1.3.1.2.23.2 IG_close

When your application no longer needs the use of ImageGear functions, you should close ImageGear using this function. This function frees resources and memory that ImageGear allocated while it was in use. This function should be called before the application program exits.

lpData must be the same value as one passed to preceding call of IG_initialize().

Declaration:

```
AT_ERRCOUNT IG_close ( LPVOID lpData);
```

Arguments:

Name	Type	Description
lpData	LPVOID	This argument currently isn’t used, and is reserved for future use. Must be set to NULL for now.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Sample:

ImageGearDemo

Example:

```
/* Close ImageGear */
nErrCount = IG_close (lpData);
```

See Also:

IG_initialize()

1.3.1.2.24 Multi Page Image File Functions

This section provides information about the Multi Page Image File group of functions.

- IG_mpf_info_get
- IG_mpf_page_count_get
- IG_mpf_page_delete
- IG_mpf_page_get
- IG_mpf_page_info_get
- IG_mpf_page_info_get_ex
- IG_mpf_page_load
- IG_mpf_page_save
- IG_mpf_page_swap
- IG_mpf_page_unload
- IG_mpf_tile_count_get

1.3.1.2.24.1 IG_mpf_info_get

If a multi-page image hMIGear is associated with the external file image, then this function returns the file format type of the external image through a second parameter, lpFileType.

Declaration:

```
AT_ERRCOUNT IG_mpf_info_get(
        [IN]  HMIGEAR hMIGear,
        [OUT] LPAT_MODE lpFileType,
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	Handle to allocated a multi-page image.
lpFileType	LPAT_MODE	Far pointer indicating where to receive file types such as IG_FORMAT_TIF, or IG_FORMAT_MODCA. For a complete list of format types, see the accucnst.h file.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR  hMIGear;  /* handle to multi-page image */
AT_ERRCOUNT      nErrCount = IGE_SUCCESS;  /* will hold returned error count  */
AT_MODE         nFileType;
 ...
/* initialize multi-page image and assign it with external file */
        iErrCnt = IG_mpf_info_get( hMIGear, &nFileType );
        if (!iErrCnt)
        printf("File type:%i\n", (INT)nFileType);
```

Remarks:

This constant is from the format constant list, which is defined in the accucnst.h include file. If it is not associated with the file, then this function returns an error.

1.3.1.2.24.2 IG_mpf_page_count_get

Uses the IG_mpi_file_open() function to get the number of pages in the external file if it is associated with multi-page image.

Declaration:

```
AT_ERRCOUNT IG_mpf_page_count_get(
        [IN] HMIGEAR hMIGear,
        [OUT] LPUINT lpnPageCount
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	Handle of the allocated multi-page image.
lpnPageCount	LPUINT	Pointer indicating where to return number of pages of the associated file.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR  hMIGear;      /* handle to multi-page image  */
AT_ERRCOUNT              nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT nPageCountI = 0; /* number of pages that should get from multi-page image */
UINT nPageCountF = 0; /* number of pages that should get from external source */
HIGEAR hIGear;      /* handle of an image */
 ...
/* initialize multi-page image and assign it with external file */
nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCountI );
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCountF );
printf("Number of pages of multi-page image is:%i\n", nPageCountI);
printf("Number of pages of external source is:%i", nPageCountF);
```

Remarks:

If the file is not associated with a multi-page image file, the value is 0.

1.3.1.2.24.3 IG_mpf_page_delete

This function deletes the nCount number of pages starting with the nStartPage.

Declaration:

```
AT_ERRCOUNT IG_mpf_page_delete(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nStartPage,
        [IN] UINT nCount
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	Handle to the allocated multi-page image.
nStartPage	UINT	The first page to be deleted.
nCount	UINT	Number of pages to delete.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR  hMIGear;       /* handle to multi-page image */
AT_ERRCOUNT      nErrCount = IGE_SUCCESS; /* will hold returned error count  */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
UINT i;
HIGEAR hIGear;      /* handle of an image */
 ...
/* initialize multi-page image and assign it with external file */
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
for ( i = 0; i < nPageCount; i++ )
        if (!nErrCount)
                if ( !IG_mpi_page_is_valid(hMIGear, i) && (!nErrCount) )
                        nErrCount = IG_mpf_page_delete( hMIGear, i, 1 );
```

Remarks:

This function then shifts pages with higher numbers to fill the space in the external file associated with the multi-page image using function IG_mpi_file_open(). The multi-page image itself is not changed. Either the multi-page image, or in the external file pages are numbered starting with 0.

Not all format filters are supported by this operation. Use the IG_fltr_info_get() function to obtain all information about the supported features for a particular format filter.

1.3.1.2.24.4 IG_mpf_page_get

If a multi-page image is associated with the external file, it loads a page of the specified index from an external file and returns it using the parameter lphIGear.

Declaration:

```
AT_ERRCOUNT IG_mpf_page_get(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nPage,
        [OUT] LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	Handle to the allocated multi-page image.
nPage	UINT	The index page to load.
lphIGear	LPHIGEAR	Indicates where to return the image handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
    AT_ERRCOUNT nErrCount; // Returned count of errors
    HMIGEAR hMPDoc;        // Handle of multipage image
    HIGEAR hIGear;         // Handle of the page image
    nErrCount = IG_mpi_create(&hMPDoc, 0);
    if(nErrCount == 0)
    {
        nErrCount = IG_mpi_file_open("multipage.tif", hMPDoc, 0, IG_MP_OPENMODE_READONLY);
        if(nErrCount == 0)
        {
            nErrCount = IG_mpf_page_get(hMPDoc, 0, &hIGear);
            if(nErrCount == 0)
            {
                //...
                // Destroy the image
                IG_image_delete(hIGear);
            }
        }
        // Destroy multipage document
        IG_mpi_delete(hMPDoc);
    }
```

Remarks:

If a multi-page image is not associated with the external file, or a failure to load a page occurs, then an error is set. This function does not change the multi-page image.

1.3.1.2.24.5 IG_mpf_page_info_get

If multi-page image hMIGear is associated with an external file, then this function returns information about the page with an nPage index from the external file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpf_info_get(
        [IN] HMIGEAR  hMIGear,
        [IN] UINT nPage,
        [OUT] LPAT_MODE lpCompression,
        [OUT] LPAT_DIB lpDib
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to allocated the multi-page image.
nPage	UINT	Index of the page, starting with 0.
lpCompression	LPAT_MODE	Far pointer indicating where to receive the compression method of the given image. It is used for constants such as IG_COMPRESSION_NONE, or IG_COMPRESSION_JPEG. A complete list of compression methods can be found in the accucnst.h file.
lpDib	LPAT_DIB	The far pointer indicating where to return a DIB associated with the page.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB – 1, 4, 8 bpp;
Grayscale – 9…16 bpp;
RGB – 24 bpp;
CMYK – 32 bpp.

This function is only kept for backward compatibility reasons. Please use IG_mpf_page_info_get_ex instead.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image  */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT nPageCount = 0;
UINT i;
AT_MODE                 nCompression;
AT_DIB          Dib;
char str[80];
 ...
/* initialize multi-page image and assign it with external file */
     nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
        for( i = 0; i < nPageCount; i++)
          if (!nErrCnt)
        {
        nErrCnt = IG_mpf_page_info_get( hMIGear, i, &nCompression, &Dib );
        if (!nErrCnt)
        {
        IG_guidlg_compression_name_get( nCompression, str, sizeof(str) );
        printf( "Page %i\n  Compression method: %s\n", i, str );
        }
}
```

Remarks:

Pages are numbered starting with 0. If the image is not associated with an external file, then this functions returns an error.

1.3.1.2.24.6 IG_mpf_page_info_get_ex

If multi-page image hMIGear is associated with an external file, then this function returns information about the page with an nPage index from the external file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpf_page_info_get_ex(
        HMIGEAR hMIGear,
        UINT nPage,
        LPAT_MODE lpCompression,
        HIGDIBINFO* lpDIB
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	Handle to allocated multi-page image.
nPage	UINT	Index of the page, starting with 0.
lpCompression	AT_MODE	Pointer indicating where to receive the compression method of the given image. It is used for constants such as IG_COMPRESSION_NONE, or IG_COMPRESSION_JPEG. A complete list of compression methods can be found in the accucnst.h file.
lpDIB	HIGDIBINFO*	Pointer indicating where to return a DIB info handle associated with the page.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image  */
AT_ERRCOUNT nErrCount; /* will hold returned error count */
UINT nPageCount = 0;
UINT i;
AT_MODE nCompression;
HIGDIBINFO hDIB;
char str[80];
...
/* initialize multi-page image, assign it with external file */
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
for( i = 0; i < nPageCount; i++)
if (!nErrCnt)
{
        nErrCnt = IG_mpf_page_info_get_ex( hMIGear, i, &nCompression, &hDIB );
        if (!nErrCnt)
        {
                IG_guidlg_compression_name_get( nCompression, str, sizeof(str) );
                printf( "Page %i\n  Compression method: %s\n", i, str );
        }
}
```

Remarks:

Pages are numbered starting with 0. If the image is not associated with an external file, then this function returns an error.

1.3.1.2.24.7 IG_mpf_page_load

If a multi-page image is associated with an external file, it loads and stores the specified number of pages from the external file into a multi-page image.

Declaration:

```
AT_ERRCOUNT IG_mpf_page_load(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nStartPage,
        [IN] UINT nCount
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle of the allocated multi-page image.
nStartPage	UINT	The first page to be loaded.
nCount	UINT	The total number of pages to be loaded.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
UINT i;
HIGEAR hIGear;     /* handle of an image */
 ...
/* initialize multi-page image and assign it with external file */
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
for ( i = 0; i < nPageCount; i++ )
        if (!nErrCount)
        if ( !IG_mpi_page_is_valid(hMIGear, i) && (!nErrCount) )
        nErrCount = IG_mpf_page_load( hMIGear, i, 1 );
```

Remarks:

To access the loaded pages, use the IG_mpi_page_get() function.

If the multi-page image is not associated with an external file, or a failure to load a page occurs, then an error is set. This function loads each Nth page from a file into the correspondent Nth page into the multi-page image. Previous page values are not deleted with function IG_image_delete(). If necessary, the number of pages is expanded to fit all loaded pages.

The access to the same PDF document from multiple threads is not permitted because the multiple threads cannot share Adobe PDF Library data types. PDF docs created/opened in the main thread can be only used from the main thread.

1.3.1.2.24.8 IG_mpf_page_save

If the multi-page image is associated with an external file then this function saves the specified pages from multi-page image to the external file.

Declaration:

```
AT_ERRCOUNT IG_mpf_page_save(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nStartPage,
        [IN] UINT nCount,
        [IN] AT_MODE nCompression,
        [IN] AT_MODE nSaveMode
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle of the multi-page image.
nStartPage	UINT	First page to save.
nCount	UINT	Total number of pages to save.
nCompression	AT_MODE	Compression method to be used when saving the pages.
nSaveMode	AT_MODE	The methods for saving the pages can be: • IG_MPF_SAVE_INSERT • IG_MPF_SAVE_REPLACE

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;      /* handle to multi-page image */
AT_ERRCOUNT             nErrCount = IGE_SUCCESS; /* will hold returned error count  */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
UINT i;
HIGEAR hIGear;      /* handle of an image */
 ...
/* initialize multi-page image and assign it with external file */
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
if ( !nErrCount && (nPageCount > 0) )
IG_mpf_page_save( hMIGear, 0, 1, IG_COMPRESSION_NONE, IG_MPF_SAVE_INSERT );
```

Remarks:

If nSaveMode is IG_MPF_SAVE_INSERT, then the specified nCount number of pages are inserted, starting with nStartPage index, and all previous pages are shifted to a higher page number. If nSaveMode is IG_MPF_SAVE_REPLACE, then the function replaces the specified nCount number of pages, starting with the nStartPage index. If a page is NULL in the multi-page image, it is skipped and not saved.

This function takes each specified Nth page from a multi-page image and saves it as the Nth page into an external file. If the image is not associated with an external file, then it returns an error.

Not all format filters support IG_MPF_SAVE_REPLACE mode. Use the IG_fltr_info_get() function to obtain information about the supported features for specific format filters.

1.3.1.2.24.9 IG_mpf_page_swap

If the multi-page image is associated with an external file, then this function swaps pages with the given page numbers in the external file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpf_page_swap(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nPage1,
        [IN] UINT nPage2
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle of the multi-page image.
nPage1	UINT	The number of first page to swap.
nPage2	UINT	The number of second page to swap.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR   hMIGear;       /* handle to multi-page image  */
AT_ERRCOUNT  nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
 ...
/* initialize multi-page image and assign it with external file */
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
/* Swap first page and last pages */
if (!nErrCount && nPageCount > 1)
        nErrCount += IG_mpf_page_swap(hMIGear, 0, nPageCount - 1 );
```

Remarks:

If the multi-page image is not associated with an external file, an error is returned. Pages are numerated starting with 0. The multi-page image is not changed.

Not all format filters support IG_mpf_page_swap() mode. Use the IG_fltr_info_get() function to obtain information about the supported features for specific format filters.

1.3.1.2.24.10 IG_mpf_page_unload

This function calls IG_image_delete() for the nCount number of pages in the multi-page image, starting with the nStartPage position.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpf_page_unload(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nStartPage,
        [IN] UINT nCount
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the valid multi-page image.
nStartPage	UINT	The number of the first page to unload.
nCount	UINT	The total number of pages to unload.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR  hMIGear;      /* handle to multi-page image */
AT_ERRCOUNT             nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
UINT i;
HIGEAR hIGear;  /* handle of an image */
 ...
/* initialize multi-page image and assign it with external file */
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
for ( i = 0; i < nPageCount; i++ )
        if ( IG_mpi_page_is_valid(hMIGear, i) && (!nErrCount) )
        nErrCount = IG_mpf_page_unload( hMIGear, i, 1 );
```

Remarks:

The number of pages in the multi-page image is not changed, but the specified positions are removed and set to a default value of NULL. The pages are numbered starting with 0.

1.3.1.2.24.11 IG_mpf_tile_count_get

If the multi-page image is associated with an external file, then this function returns the tile information for the specified page number.

Declaration:

```
AT_ERRCOUNT IG_mpf_tile_count_get(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nPage,
        [OUT] LPUINT lpTileRows,
        [OUT] LPUINT lpTileCols
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle of the multi-page image.
nPage	UINT	The page number from which to get the tile count.
LpTileRows	LPUINT	Pointer to a UINT variable to receive the number of tiles horizontally (number of tiles in a row).
LpTileCols	LPUINT	Pointer to a UINT variable to receive the number of tiles vertically (number of tiles in a column).

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR  hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT             nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
UINT nTileRows;
UINT nTileCols;
 ...
/* initialize multi-page image and assign it with external file */
nErrCount = IG_mpf_page_count_get( hMIGear, &nPageCount );
/* Count number of tiles in the first page */
if (!nErrCount && nPageCount > 0)
        nErrCount += IG_mpf_tile_count_get(hMIGear, 0, &nTileRows, &nTileCols );
if (!nErrCount)
{
        printf( "Number of tiles in a row:%i\n", nTileRows );
        printf( "Number of tiles in a colomns:&i\n", nTileCols );
}
```

Remarks:

An error is returned if the image is not associated with an external file. Pages are numbered starting with 0.

1.3.1.2.25 Multi Page Image Functions

This section provides information about the Multi Page Image group of functions.

- IG_mpi_CB_get
- IG_mpi_CB_reset
- IG_mpi_CB_reset_all
- IG_mpi_CB_set
- IG_mpi_close
- IG_mpi_create
- IG_mpi_delete
- IG_mpi_file_open
- IG_mpi_file_save
- IG_mpi_info_get
- IG_mpi_is_valid
- IG_mpi_page_count_get
- IG_mpi_page_count_set
- IG_mpi_page_delete
- IG_mpi_page_get
- IG_mpi_page_is_valid
- IG_mpi_page_set

1.3.1.2.25.1 IG_mpi_CB_get

This function returns information about the associated private data pointer and update function using the dwCBID identifier.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_CB_get(
        [IN] HMIGEAR hMIGear,
        [IN] DWORD dwCBID,
        [OUT] LPVOID FAR* lplpPrivate,
        [OUT] LPFNIG_MPCB_UPDATE FAR* lplpfnUpdate
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle of the allocated multi-page image.
dwCBID	DWORD	A unique identifier of the private data and function.
lplpPrivate	LPVOID FAR*	A pointer indicating where to receive the private data.
lplpfnUpdate	LPFNIG_MPCB_UPDATE FAR*	A pointer indicating where to receive the update function.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR          hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error count */
LPVOID lpData;
LPFNIG_MPCB_UPDATE        lpUpdateFunc;
DWORD dwCBID;
 ...
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear, _MPWndUpdate, &dwCBID );
 ...
nErrCount = IG_mpi_CB_get( hMIGear, dwCBID, &lpData, &lpUpdateFunc );
 ...
VOID  ACCUAPI  _MPWndUpdate(
   DWORD          dwCBID,
   LPVOID         lpPrivate,     /* Private data passed in */
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
)
{
   switch( nMode )
   {
   case  IG_MPCBMODE_MPI_DELETE:
         ...
      break;
   case  IG_MPCBMODE_MPI_ASSOCIATED:
         ...
      break;
```

```
    case  IG_MPCBMODE_MPI_CLOSE:
          ...
       break;
          ...
    }
 }
```

Remarks:

See the IG_mpi_CB_set() documentation for a description of how notification works with multi-page images.

If there is no association with an external file, then NULL values are assigned to both pointers.

1.3.1.2.25.2 IG_mpi_CB_reset

This function removes previously associated callback data from the multi-page image using the dwCBID identifier.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_CB_reset(
        [IN] HMIGEAR hMIGear,
        [IN] DWORD dwCBID
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	Handle to allocated multi-page image.
dwCBID	DWORD	Unique identifier of private data and function.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR  hMIGear;      /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error count  */
LPVOID lpData;
LPFNIG_MPCB_UPDATE    lpUpdateFunc;
DWORD dwCBID;
 ...
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear, _MPWndUpdate, &dwCBID );
 ...
nErrCount = IG_mpi_CB_reset( hMIGear, dwCBID );
 ...
}
VOID  ACCUAPI  _MPWndUpdate(
   DWORD          dwCBID,
   LPVOID         lpPrivate,     /* Private data passed in        */
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
)
{
   switch( nMode )
   {
   case  IG_MPCBMODE_MPI_DELETE:
         ...
      break;
   case  IG_MPCBMODE_MPI_ASSOCIATED:
         ...
      break;
   case  IG_MPCBMODE_MPI_CLOSE:
         ...
      break;
         ...
   }
```

```
}
```

Remarks:

See the IG_mpi_CB_set()documentation for a description of how notification works with multi-page images. After calling this function, the appropriate callback function receives notifications and removes the data from the active list.

1.3.1.2.25.3 IG_mpi_CB_reset_all

This function works the same way as the IG_mpi_CB_reset() function, but removes all callback data and functions from the multi-page image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_CB_reset_all(
        [IN] HMIGEAR hMIGear
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the allocated multi-page image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;      /* handle to multi-page image */
AT_ERRCOUNT              nErrCount = IGE_SUCCESS; /* will hold returned error count   */
DWORD dwCBID1, dwCBID2;
 ...
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear, _MPWndUpdate, &dwCBID1 );
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear, _MPWndUpdate, &dwCBID2 );
 ...
nErrCount = IG_mpi_CB_reset_all( hMIGear );
 ...
}
VOID  ACCUAPI  _MPWndUpdate(
   DWORD          dwCBID,
   LPVOID         lpPrivate,     /* Private data passed in  */
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
)
{
   switch( nMode )
   {
   case  IG_MPCBMODE_MPI_DELETE:
         ...
      break;
   case  IG_MPCBMODE_MPI_ASSOCIATED:
         ...
      break;
   case  IG_MPCBMODE_MPI_CLOSE:
         ...
      break;
         ...
   }
}
```

1.3.1.2.25.4 IG_mpi_CB_set

Use this function to call code that associates the given multi-page image hMIGear with any lpPrivate data, and updates the defined function.

Declaration:

```
AT_ERRCOUNT ACCUAPI  IG_mpi_CB_set(
        [IN] HMIGEAR hMIGear,
        [IN] LPVOID lpPrivate,
        [IN] LPFNIG_MPCB_UPDATE lpfnUpdate,
        [OUT] LPDWORD lpdwCBID,
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the allocated multi-page image.
lpPrivate	LPVOID	Any private data.
lpfnUpdate	LPFNIG_MPCB_UPDATE	The pointer to the update function.
lpdwCBID	LPDWORD	Indicates where to return the ID of the associated callback data.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT  nErrCount = IGE_SUCCESS; /* will hold returned error count */
LPVOID lpData;
LPFNIG_MPCB_UPDATE    lpUpdateFunc;
DWORD dwCBID;
 ...
nErrCount = IG_mpi_CB_set( hMIGear, (LPVOID)hMIGear, _MPWndUpdate, &dwCBID );
 ...
nErrCount = IG_mpi_CB_get( hMIGear, dwCBID, &lpData, &lpUpdateFunc );
 ...
}
VOID  ACCUAPI  _MPWndUpdate(
   DWORD          dwCBID,
   LPVOID         lpPrivate,     /* Private data passed in */
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
)
{
   switch( nMode )
   {
   case  IG_MPCBMODE_MPI_DELETE:
         ...
      break;
   case  IG_MPCBMODE_MPI_ASSOCIATED:
         ...
```

```
        break;
    case  IG_MPCBMODE_MPI_CLOSE:
            ...
        break;
            ...
    }
}
```

Remarks:

Multi-page images allow you to notify the application about status changes. Use this function to call code that associates the given multi-page image hMIGear with any lpPrivate data, and updates the defined function. See LPFNIG_MPCB_UPDATE.

1.3.1.2.25.5 IG_mpi_close

If the multi-page image hMIGear was previously associated with an external image file using the function IG_mpi_file_open(), then this function closes the file and frees all corresponding resources; if the multi-page image is not associated with an external file, then this function does nothing.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_close( HMIGEAR hMIGear );
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the allocated multi-page image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS;  /* will hold returned error count */
nErrCount = IG_mpi_create( &hMIGear, 0 );
 ...
/* any operations with hMIGear */
nErrCount = IG_mpi_close( hMIGear );
```

1.3.1.2.25.6 IG_mpi_create

This function allocates and initializes a new multi-page image and returns its handle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_create(
        LPHMIGEAR lphMIGear,
        UINT nPages
);
```

Arguments:

Name	Type	Description
lpHMIGear	LPHMIGEAR	A pointer indicating where to return the handle of the allocated and initialized multi-page image.
nPage	UINT	The number of multi-page image pages that should be created.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR       hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT   nErrCount = IGE_SUCCESS; /* will hold returned error count */
nErrCount = IG_mpi_create( &hMIGear, 0 );
 ...
/* any operations with hMIGear */
nErrCount = IG_mpi_delete( hMIGear );
```

Remarks:

This new image is set with nPages. Each image is initialized with the default value NULL. If there is a failure, then the returned handle is NULL and an error is set.

The multi-page image is array of pages where each page is a HIGEAR object. All pages are numbered beginning with a 0 index, so that 0 - is the first page, 1 - is the second page, etc. If the image contains nCount number of pages, then its pages can be accessed through indexes 0 - nCount-1. The value of each page can be either NULL (default value) or value HIGEAR image.

1.3.1.2.25.7 IG_mpi_delete

This function calls IG_image_delete() for all valid pages of a multi-page image and frees all resources allocated with multi-page image handle.

Declaration:

```
AT_ERRCOUNT ACCUAPI  IG_mpi_delete( HMIGEAR hMIGear );
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	Handle to allocated multi-page image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT             nErrCount = IGE_SUCCESS; /* will hold returned error count */
nErrCount = IG_mpi_create( &hMIGear, 0 );
 ...
/* any operations with hMIGear */
nErrCount = IG_mpi_delete( hMIGear );
```

Remarks:

If it is associated with external file then it is closed with IG_mpi_close() before deletion.

1.3.1.2.25.8 IG_mpi_file_open

This function allows you to associate a multi-page image with an external image file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_file_open(
        [IN] const LPCHAR lpFileName,
        [IN] HMIGEAR hMIGear,
        [IN] AT_MODE nFormat,
        [IN] AT_MODE nOpenMode
);
```

Arguments:

Name	Type	Description
lpFileName	const LPCHAR	The far pointer to the filename (you may include the path with the filename) of the image file to be associated with a given multi-page image.
hMIGear	HMIGEAR	The handle to the allocated multi-page image.
nFormat	AT_MODE	The format of the file, such as IG_FORMAT_UNKNOWN or IG_FORMAT_TIF. See the accucnst.h file for a list of all available IG_FORMAT_... constants.
nOpenMode	AT_MODE	An AT_MODE constant, such as IG_MP_OPENMODE_READONLY or IG_MP_OPENMODE_READWRITE.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR  hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT             nErrCount = IGE_SUCCESS; /* will hold returned error count */
nErrCount = IG_mpi_create( &hMIGear, 0 );
if (!nErrCount)
nErrCount = IG_mpi_file_open( "picture1.tif", hMIGear, 0, IG_MP_OPEN_READ );
...
nErrCount = IG_mpi_delete( hMIGear );
```

Remarks:

After the association is made, you can then use different page manipulation functions, such as page load, save, delete, and swap. With this association operation, ImageGear allows you to store internal data, allowing you to make page operations faster than if using IG_fltr_... functions. This file can be opened with two modes - IG_MP_OPENMODE_READONLY and IG_MP_OPENMODE_READWRITE.

The first mode, read-only access, it is used only when page loading is necessary. It does not allow you to change the external file. When the image is opened with read only access, it sets the number of pages in the multi-page image equal to the number of pages in the external file using IG_mpi_page_count_set() function.

The second mode opens file for read-write access and allows all possible page operations with the external file. The multi-page image is not changed.

Not all filters support all page manipulation operations. Use the function IG_fltr_info_get(), which returns the information about all supported features of a particular filter.

IG_MP_OPENMODE_NONE is also accepted as a value for nOpenMode and, in this case, this call is equivalent to the

IG_mpi_close() call.

The nFormat parameter is used only if a new image file is to be created and nOpenMode= IG_MP_OPENMODE_READWRITE. In this case, the file of the specified format is created. In all other cases this parameter is ignored.

1.3.1.2.25.9 IG_mpi_file_save

Use this function to save a multi-page vector document to a file.

Declaration:

```
AT_ERRCOUNT LACCUAPI IG_mpi_file_save(
        [IN] const LPCHAR lpFileName,
        [IN] HMIGEAR hMIGear,
        [IN] UINT nStartPageFile,
        [IN] UINT nStartIndexDoc,
        [IN] UINT nPageCount,
        [IN] AT_MODE nFormat,
        [IN] AT_MODE nSaveMode
);
```

Arguments:

Name	Type	Description
lpFileName	const LPCHAR	Output file name.
hMIGear	HMIGEAR	Multi-page vector document.
nStartPageFile	UINT	When nSaveMode is IG_MPI_SAVE_APPEND, specifies the page number in the output file after which pages from hMIGear are inserted. The first page is 0. Set nStartPageFile to -1 to append pages after the last page of the existing document.
nStartIndexDoc	UINT	The page number of the first page in hMIGear to save. The first page is 0.
nPageCount	UINT	Total number of pages to save.
nFormat	AT_MODE	IG_FORMAT_PDF or IG_FORMAT_POSTSCRIPT.
nSaveMode	AT_MODE	IG_MPI_SAVE_OVERWRITE or IG_MPI_SAVE_APPEND.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear; /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT pageCount;
nErrCount = IG_mpi_create( &hMIGear, 0 );

if (!nErrCount)
nErrCount = IG_mpi_file_open( "input.pdf", hMIGear, 0, IG_MP_OPENMODE_READWRITE );

nErrCount = IG_mpi_page_count_get(hMIGear, &pageCount);

if (!nErrCount)
nErrCount = IG_mpi_file_save("output.pdf", hMIGear, -1, 0, pageCount, IG_FORMAT_PDF,
IG_MPI_SAVE_OVERWRITE);

nErrCount = IG_mpi_delete( hMIGear );
```

Remarks:

This function is only used for multi-page vector documents. The following formats are currently supported by this API: PDF, PostScript.

Two saving modes are currently supported: IG_MPI_SAVE_OVERWRITE and IG_MPI_SAVE_APPEND. These modes define how to process pages if lpFileName points to an existing file of the same type. If no file exists with a given file name, then nStartPageFile and nSaveMode are ignored, and the function saves pages to a new file.

IG_MPI_SAVE_OVERWRITE means that all pages in existing file should be removed, and pages from hMIGear should be placed instead. In this mode nStartPageFile parameter is ignored, because no original pages are left in the file.

IG_MPI_SAVE_APPEND means that pages from hMIGear should be either appended or inserted into the document, depending on nStartPageFile parameter.

In both save modes, if nStartIndexDoc is out of hMIGear pages range, or nStartIndexDoc + nPageCount is out of hMIGear pages range, a "Bad Parameter" error is thrown.

1.3.1.2.25.10 IG_mpi_info_get

This function returns status information for the multi-page image, such as the association type, open mode, as well as others.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_info_get(
        [IN] HMIGEAR hMIGear,
        [IN] AT_MODE nMode,
        [IN/OUT] LPVOID lpData,
        [IN] DWORD dwSize
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the allocated multi-page image.
nMode	AT_MODE	This argument is used to determine the type of information being retrieved.
lpData	LPVOID	The far pointer to the buffer indicating where to return the allocated data.
dwSize	DWORD	The size of the buffer.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR  hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT      nErrCount = IGE_SUCCESS;  /* will hold returned error count */
HIGEAR hIGear;      /* handle of an image */
AT_MODE nAType;
CHAR FileName[_MAX_PATH];
 ...
/* initialize of multi-page image and assign it with external file */
nErrCount = IG_mpi_info_get( hMIGear, IG_MP_ASSOCIATION_TYPE , &nAType, sizeof(nAType) );
if( nAType== IG_MP_ASSOCIATE_FILE )
{
        IG_mpi_info_get( hMIGear, IG_MP_FILE_NAME, FileName, sizeof(FileName) );
}
```

Remarks:

The value of the parameters depends on nMode. The following table lists the possible combinations:

AT_MODE Constants	Type of Third Argument	dwSize	Description
IG_MP_ASSOCIATION_TYPE	LPAT_MODE	Sizeof (AT_MODE)	Returns the association type of the given multi-page image, which can be: ● IG_MP_ASSOCIATE_NONE ● IG_MP_ASSOCIATE_FILE ● IG_MP_ASSOCIATE_MEMORY [not

			currently implemented]
IG_MP_OPENMODE	LPAT_MODE	Sizeof(AT_MODE)	Returns the open mode of the associated file, which can be: ● IG_MP_OPENMODE_NONE ● IG_MP_OPENMODE_READONLY ● IG_MP_OPENMODE_READWRITE
IG_MP_FILE_NAME	LPCHAR	Length of the buffer including last 0 byte	Returns the name of the associated file.
[*]IG_MP_MEM_BUFFER_PTR	LPBYTE	4	Returns a pointer to the memory associated with the multi-page image.
[*]IG_MP_MEM_BUFFER_SIZE	LPDWORD	4	Returns the size of the associated memory.
IG_MP_FORMAT			File format of the multi-page document. One of the enumIGFormats values.
IG_MP_DOCUMENT			Native document associated with the multi-page document.

1.3.1.2.25.11 IG_mpi_is_valid

This function is used for checking whether the hMIGear value is a valid multi-page image handle.

Declaration:

```
AT_BOOL ACCUAPI IG_mpi_is_valid( HMIGEAR hMIGear );
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	Handle to allocated multi-page image.

Return Value:

Returns TRUE if the given multi-page image is valid.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;       /* handle to multi-page image */
AT_ERRCOUNT             nErrCount = IGE_SUCCESS; /* will hold returned error count */
nErrCount = IG_mpi_create( &hMIGear, 0 );
if ( IG_mpi_is_valid(hMIGear) )
        printf("The multi-page image is valid!");
else
        printf("It's not valid multi-page image!");
IG_mpi_delete( hMIGear );
```

Remarks:

The multi-page image is valid if the IG_mpi_create() function is returned successfully and IG_mpi_delete() has not been called.

1.3.1.2.25.12 IG_mpi_page_count_get

This function returns the number of pages in a multi-page image (the size of page array).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_page_count_get(
        [IN] HMIGEAR hMIGear,
        [OUT] LPUINT lpPageCount
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	Handle to the allocated multi-page image.
lpPageCount	LPUINT	Pointer indicting where to return the number of pages of the given multi-page image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;    /* handle to multi-page image */
AT_ERRCOUNT             nErrCount = IGE_SUCCESS; /* will hold returned error count  */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
UINT i,
                j = 0;
HIGEAR hIGear;    /* handle of an image */
 ...
/* initialize multi-page image and assign it with external file */
nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );
for ( i = 0; i < nPageCount; i++ )
        if (!nErrCount)
        {
                nErrCount = IG_mpi_page_get( hMIGear, i, &hIGear );
                if (IG_mpi_page_is_valid(hMIGear, i))
                        j++;
        }
printf("Number of valid pages is:%i", j);
```

Remarks:

The HIGEAR handle of any given page is not significant in this function, other than to identify the image; this function simply counts the array size.

1.3.1.2.25.13 IG_mpi_page_count_set

This function sets the size of the page array of a valid multi-page image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_page_count_set(
        [IN] HMIGEAR  hMIGear,
        [IN] UINT  nPageCount
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the allocated multi-page image.
nPageCount	UINT	The new number of pages.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;       /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
UINT i,
 ...
/* initialize of multi-page image and assign it with external file */
nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );
if (!nErrCount)
{
        i = 0;
        while( !IG_mpi_page_is_valid(hMIGear, nPageCount - i - 1 ) && (i < nPageCount) )
                i++;
        }
        if (!nErrCount)
                nErrCount = IG_mpi_page_count_set( hMIGear, nPageCount - i );
}
```

Remarks:

If the size is increased, then new pages are initialized with a default value of NULL. If the array is reduced, then the removed pages are not deleted with the function IG_image_delete(). This function applies only to the image, but does not affect the associated file. The number of pages is not changed.

1.3.1.2.25.14 IG_mpi_page_delete

This function deletes the nCount number of elements starting with the nStartPage index from the page array of the multi-page image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mpi_page_delete(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nStartPage,
        [IN] UINT nCount
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the allocated multi-page image.
nStartPage	UINT	The first page to delete.
nCount	UINT	The total number of pages to delete.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;     /* handle to multi-page image */
AT_ERRCOUNT nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
UINT i;
HIGEAR hIGear;         /* handle of an image */
 ...
/* initialize of multi-page image and assign it with external file */
nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );
/* delete all pages and shrink array to entries pages */
nErrCount = IG_mpi_page_delete( hMIGear, 0, nPageCount );
```

Remarks:

If the deleted pages are valid images, they are deleted with the function IG_image_delete(). Pages with higher indexes are shifted by removing the number and number of pages is decreased so that all pages are numbered from 0 to nPageCount-1.

For multi-page vector documents, this function DELETES a page in the underneath vector document data. The following formats are currently supported by this API: PDF, PostScript.

1.3.1.2.25.15 IG_mpi_page_get

This function returns the value of a page array with index nPage.

Declaration:

```
AT_ERRCOUNT  ACCUAPI  IG_mpi_page_get(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nPage,
        [OUT] LPHIGEAR lpHIPage
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the allocated multi-page image.
nPage	UINT	The number of the page to return.
lpHIPage	LPHIGEAR	The far pointer indicating where to return a handle of the page.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR hMIGear;        /* handle to multi-page image */
AT_ERRCOUNT             nErrCount = IGE_SUCCESS;   /* will hold returned error count */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
UINT I = 0;
HIGEAR hIGear;     /* handle of an image */
 ...
nErrCount = IG_mpi_page_get( hMIGear, i, &hIGear );
 ...
```

Remarks:

This value can be either NULL or a valid HIGEAR handle.

The multi-page image is not changed. This function makes a copy of the page handle.

This function does not load the page from an external file, but just returns the current page value of the multi-page image.

For multi-page vector documents, this function GETS a page from the underneath vector document data. The following formats are currently supported by this API: PDF, PostScript.

1.3.1.2.25.16 IG_mpi_page_is_valid

This function returns information about the page with a given index.

Declaration:

```
AT_BOOL ACCUAPI IG_mpi_page_is_valid(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nPage,
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the allocated multi-page image.
nPage	UINT	The index of the page to check.

Return Value:

Returns TRUE if the given page is valid.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR   hMIGear;      /* handle to multi-page image */
AT_ERRCOUNT             nErrCount;   /* will hold returned error count */
UINT nPageCount = IGE_SUCCESS; /* number of pages that should get from multi-page image */
UINT i;
HIGEAR hIGear;     /* handle of an image */
 ...
/* initialize of multi-page image and assign it with external file */
nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );
for ( i = 0; i < nPageCount; i++ )
        if ( !IG_mpi_page_is_valid(hMIGear, i) && (!nErrCount) )
        nErrCount = IG_mpi_page_unload( hMIGear, i, 1 );
```

Remarks:

If the page with such an index exists as well as a valid HIGEAR handle, then it returns TRUE. In all other cases FALSE is returned.

1.3.1.2.25.17 IG_mpi_page_set

This function assigns a new value to the page of the multi-page image.

Declaration:

```
AT_ERRCOUNT IG_mpi_page_set(
        [IN] HMIGEAR hMIGear,
        [IN] UINT nPage,
        [IN] HIGEAR hIPage
);
```

Arguments:

Name	Type	Description
hMIGear	HMIGEAR	The handle to the allocated multi-page image.
nPage	UINT	The index of the page to set.
hIPage	HIGEAR	The handle of the image to set.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HMIGEAR   hMIGear;       /* handle to multi-page image */
AT_ERRCOUNT             nErrCount = IGE_SUCCESS; /* will hold returned error count */
UINT nPageCount = 0; /* number of pages that should get from multi-page image */
HIGEAR hPage1,
                hPage2;
 ...
/* initialize of multi-page image and assign it with external file */
nErrCount = IG_mpi_page_count_get( hMIGear, &nPageCount );
/* Swap first page and last page */
if (!nErrCount && nPageCount > 1)
{
        nErrCount += IG_mpi_page_get( hMIGear, 0, &hPage1 );
        nErrConut += IG_mpi_page_get( hMIGear, nPageCount - 1, &hPage2 );
        if (!nErrCount)
        {
        nErrCount += IG_mpi_page_set(hMIGear, iPage1, hPage2 );
        nErrCount += IG_mpi_page_set(hMIGear, iPage2, hPage1 );
        }
}
```

Remarks:

The previous value is not deleted with the IG_image_delete() function. The size of the multi-page image is not changed, so that page arrays is not expanded when nPage is greater than nPageCount-1.

For multi-page vector documents, this function SETS a page to the underneath vector document data. The following formats are currently supported by this API: PDF, PostScript.

1.3.1.2.26 Multimedia Functions

This section provides information about the Multimedia group of functions.

- IG_mult_audio_format_get
- IG_mult_audio_format_set
- IG_mult_audio_get
- IG_mult_audio_seek_time
- IG_mult_close
- IG_mult_current_frame_advance
- IG_mult_current_frame_duration_get
- IG_mult_current_frame_image_get
- IG_mult_current_frame_info_get
- IG_mult_current_frame_info_set
- IG_mult_current_frame_is_valid
- IG_mult_current_frame_reset
- IG_mult_current_frame_seek
- IG_mult_current_frame_seek_time
- IG_mult_duration_get
- IG_mult_frame_duration_get
- IG_mult_frame_image_get
- IG_mult_frame_info_get
- IG_mult_frame_info_set
- IG_mult_frame_num_from_time_get
- IG_mult_has_audio
- IG_mult_info_get
- IG_mult_has_video
- IG_mult_info_set
- IG_mult_open_FD
- IG_mult_open_FD_format
- IG_mult_open_file
- IG_mult_open_file_format
- IG_mult_open_mem
- IG_mult_open_mem_format

1.3.1.2.26.1 IG_mult_audio_format_get

This function retrieves the format of the audio in a multimedia instance.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_audio_format_get(
        HIGMULT hMult,
        LPAT_UINT lpSampleRate,
        LPAT_UINT lpBitsPerSample,
        LPAT_UINT lpChannels
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
lpSampleRate	LPAT_UINT	Number of samples per second.
lpBitsPerSample	LPAT_UINT	Number of bits per sample.
lpChannels	LPAT_UINT	Number of channels (1 = mono, 2 = stereo).

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_UINT nSampleRate;    /* Sample rate of audio */
AT_UINT nBitsPerSample; /* Bits per sample */
AT_UINT nChannels;      /* Number of channels */
nErrcount = IG_mult_audio_format_get(hMult, &nSampleRate, &nBitsPerSample, &nChannels);
```

Remarks:

Note that audio retrieved using IG_mult_audio_get() is always in uncompressed PCM format. If the multimedia instance does not contain audio, an error will occur.

1.3.1.2.26.2 IG_mult_audio_format_set

This function sets the format of the audio to be retrieved from a multimedia instance.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_audio_format_set(
        HIGMULT hMult,
        LPAT_UINT lpSampleRate,
        LPAT_UINT lpBitsPerSample,
        LPAT_UINT lpChannels
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
lpSampleRate	LPAT_UINT	Number of samples per second.
lpBitsPerSample	LPAT_UINT	Number of bits per sample.
lpChannels	LPAT_UINT	Number of channels (1 = mono, 2 = stereo).

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Ask for 44100Hz 16-bit stereo audio */
AT_ERRCOUNT nErrcount;       /* Number of errors on stack */
HIGMULT hMult;               /* Multimedia instance handle */
AT_UINT nSampleRate = 44100; /* Sample rate of audio */
AT_UINT nBitsPerSample = 16; /* Bits per sample */
AT_UINT nChannels = 2;       /* Number of channels */
nErrcount = IG_mult_audio_format_set(hMult, &nSampleRate, &nBitsPerSample, &nChannels);
```

Remarks:

This is not a necessary step under normal circumstances. If possible, the audio will be converted to the format you specify as it is retrieved using IG_mult_audio_get(). If conversion is not possible, an error will occur.

1.3.1.2.26.3 IG_mult_audio_get

This function retrieves the next available audio data from a multimedia instance.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_audio_get(
        HIGMULT hMult,
        LPAT_VOID lpBuffer,
        LPAT_UINT lpBufferSize
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
lpBuffer	LPAT_VOID	Buffer in which to store audio data.
lpBufferSize	LPAT_UINT	Pointer to the number of bytes of audio to retrieve (in), and the number of bytes actually retrieved (out).

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_UINT nSampleRate;    /* Sample rate of audio */
AT_UINT nBitsPerSample; /* Bits per sample */
AT_UINT nChannels;      /* Number of channels */
LPAT_BYTE lpBuffer;     /* Audio buffer */
AT_UINT bufSize;        /* Size of audio buffer */
nErrcount = IG_mult_audio_format_get(hMult, &nSampleRate, &nBitsPerSample, &nChannels);
/* Calculate # of bytes for one second of audio */
bufSize = nSamplesPerSec * (nBitsPerSample / 8) * nChannels;
lpBuffer = (LPAT_BYTE) malloc(bufSize);
nErrcount = IG_mult_audio_get(hMult, lpBuffer, &bufSize);
```

Remarks:

You must specify the number of bytes of audio data to retrieve in the value pointed to by lpBufferSize, and this number must be a multiple of the sample size ((nBitsPerSample / 8) * nChannels). Audio data is provided in uncompressed PCM format.

You can call this function repeatedly for buffered access to the audio data (for example, one second of audio at a time) or you can retrieve all of the audio at once.

Note that the number of bytes retrieved by this function is returned in the lpBufferSize parameter. If you requested more data than is available (for example, you are near the end of the file) this number will be less than what you requested.

1.3.1.2.26.4 IG_mult_audio_seek_time

This function sets the starting position within a multimedia instance for the next audio to be retrieved using IG_mult_audio_get().

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_audio_seek_time(
        HIGMULT hMult,
        AT_UINT msTime
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
msTime	AT_UINT	Time in milliseconds to which to seek.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_UINT msDuration;     /* Duration of file in milliseconds */
/* Seek to beginning */
nErrcount = IG_mult_audio_seek_time(hMult, 0);
/* Seek to half way */
nErrcount = IG_mult_duration_get(hMult, NULL, &msDuration);
nErrcount = IG_mult_audio_seek_time(hMult, msDuration / 2);
```

Remarks:

If you are streaming audio by calling IG_mult_audio_get() repeatedly, you should not call this function each time you call IG_mult_audio_get(). Excessive seeking could result in discontinuities in the retrieved audio.

1.3.1.2.26.5 IG_mult_close

This function closes a multimedia instance.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_close(HIGMULT hMult);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
nErrcount = IG_mult_close(hMult);
```

Remarks:

You must call this function after opening a multimedia instance in order to ensure that all associated resources are freed.

1.3.1.2.26.6 IG_mult_current_frame_advance

This function advances the current frame to the next frame.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_current_frame_advance(HIGMULT hMult);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_BOOL bValid;         /* Is current frame valid? */
HIGEAR hIGear;          /* HIGEAR handle of image */
while (IG_mult_current_frame_is_valid(hMult, &bValid) == IGE_SUCCESS && bValid)
{
        nErrcount = IG_mult_current_frame_image_get(hMult, &hIGear);
        nErrcount = IG_mult_current_frame_advance(hMult);
}
```

Remarks:

You can use this in a loop to iterate over all of the frames. This function will not generate an error when you advance beyond the last available frame. You should use IG_mult_current_frame_is_valid() to determine whether or not the end has been reached.

1.3.1.2.26.7 IG_mult_current_frame_duration_get

This function returns the duration of the current frame in 100 nanosecond units.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_current_frame_duration_get(
        HIGMULT hMult,
        LPAT_UINT lpDuration
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
lpDuration	LPAT_UINT	Frame duration (in 100ns units).

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_UINT msDuration;     /* Duration of frame in milliseconds */
nErrcount = IG_mult_current_frame_duration_get(hMult, &duration);
```

Remarks:

This provides a reasonable level of accuracy for the short frame durations that are typical in video. To convert to milliseconds, divide by 10000.

Note that retrieving the durations of individual frames is essential for proper timing when the frame rate is variable, as is often the case with animated GIF files.

1.3.1.2.26.8 IG_mult_current_frame_image_get

This function retrieves the HIGEAR of the current frame's image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_current_frame_image_get(
        HIGMULT hMult,
        LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
lphIGear	LPHIGEAR	Image handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Saves first frame as a 128-pixel-wide thumbnail */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
HIGEAR hIGear;          /* HIGEAR handle of image */
HIGEAR hIGearCopy;      /* HIGEAR handle of duplicate */
AT_DIMENSION w, h;      /* Width and height of image */
nErrcount = IG_mult_current_frame_reset(hMult);
nErrcount = IG_mult_current_frame_image_get(hMult, &hIGear);
nErrcount = IG_image_duplicate(hIGear, &hIGearCopy);
nErrcount = IG_image_dimensions_get(hIGearCopy, &w, &h, NULL);
nErrcount = IG_IP_resize(hIGearCopy, 128, (AT_DIMENSION) ((float) h / (float) w * 128),
IG_INTERPOLATION_BICUBIC);
nErrcount = IG_save_file(hIGearCopy, "thumb.jpg", IG_SAVE_JPG);
nErrcount = IG_image_delete(hIGearCopy);
```

Remarks:

This image is owned by ImageGear. You can perform read-only operations such as display or save to a file, but do not attempt to free or alter the image. If you need to alter the image (resize it, for example), use IG_image_duplicate() to make a copy of the image, work with the copy, and free it with IG_image_delete() when finished.

1.3.1.2.26.9 IG_mult_current_frame_info_get

This function retrieves a specific piece of information about the current frame.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_current_frame_info_get(
        HIGMULT hMult,
        AT_MODE infoID,
        LPAT_VOID lpInfo,
        LPAT_UINT lpInfoSize
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
infoID	AT_MODE	ID of info to get.
lpInfo	LPAT_VOID	Pointer to buffer in which to store info.
lpInfoSize	LPAT_UINT	Size of buffer pointed to by lpInfo.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

You can specify the information to retrieve using the infoID parameter, whose value is a member of enumIGMultInfo defined in accucnst.h.

1.3.1.2.26.10 IG_mult_current_frame_info_set

This function sets a specific piece of information about the current frame.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_current_frame_info_set(
        HIGMULT hMult,
        AT_MODE infoID,
        LPAT_VOID lpInfo,
        LPAT_UINT lpInfoSize
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
infoID	AT_MODE	ID of info to set.
lpInfo	LPAT_VOID	Pointer to buffer containing the info.
lpInfoSize	LPAT_UINT	Size of buffer pointed to by lpInfo.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Specify the information to retrieve using the infoID parameter, whose value is a member of enumIGMultInfo defined in accucnst.h.

1.3.1.2.26.11 IG_mult_current_frame_is_valid

This function lets you tell whether or not the current frame is valid.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_current_frame_is_valid(
        HIGMULT hMult,
        LPAT_BOOL lpbValid
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
lpbValid	LPAT_BOOL	Boolean that is set to TRUE if the current frame is valid.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
HIGEAR hIGear;          /* HIGEAR handle of image */
AT_BOOL bValid;         /* Is current frame valid? */
while (IG_mult_current_frame_is_valid(hMult, &bValid) == IGE_SUCCESS && bValid)
{
        nErrcount = IG_mult_current_frame_image_get(hMult, &hIGear);
        nErrcount = IG_mult_current_frame_advance(hMult);
}
```

Remarks:

You can use this function in a loop over all of the frames to determine when you have advanced beyond the last available frame.

1.3.1.2.26.12 IG_mult_current_frame_reset

This function resets the current frame to the first frame.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_current_frame_reset(HIGMULT hMult);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
nErrcount = IG_mult_current_frame_reset(hMult);
```

Remarks:

You can use it to seek to the beginning of a multimedia instance.

1.3.1.2.26.13 IG_mult_current_frame_seek

This function seeks to the given frame number in a multimedia instance.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_current_frame_seek(
        HIGMULT hMult,
        AT_UINT frameNum
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
frameNum	AT_UINT	Frame number to seek to (0 = first frame).

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Seek to last frame */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
AT_UINT numFrames;      /* Total number of frames */
nErrcount = IG_mult_duration_get(hMult, &numFrames, NULL);
nErrcount = IG_mult_current_frame_seek(hMult, numFrames - 1);
```

1.3.1.2.26.14 IG_mult_current_frame_seek_time

This function seeks to the given absolute time in milliseconds since the beginning of the multimedia instance.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_current_frame_seek_time(
        HIGMULT hMult,
        AT_UINT msTime
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
msTime	AT_UINT	Time to seek to in milliseconds.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Grab frames at two second intervals */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_BOOL bValid;         /* Is the current frame valid? */
AT_UINT msTime = 0;     /* Timestamp used for seeking */
HIGEAR hIGear;          /* HIGEAR handle of image */
IG_mult_current_frame_reset(hMult);
while (IG_mult_current_frame_is_valid(hMult, &bValid) == IGE_SUCCESS && bValid)
{
        nErrcount = IG_mult_current_frame_image_get(hMult, &hIGear);
        msTime += 2000;
        nErrcount = IG_mult_current_frame_seek_time(hMult, msTime);
}
```

Remarks:

This function locates the frame that would be visible at this time under normal speed playback conditions and makes this the current frame. Seeking to a time of 0 will seek to the beginning, 1000 will seek to one second after the beginning, and so on.

1.3.1.2.26.15 IG_mult_duration_get

This function retrieves the duration of a multimedia instance as a number of frames and a duration in milliseconds.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_duration_get(
        HIGMULT hMult,
        LPAT_UINT lpNumFrames,
        LPAT_UINT lpDuration
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
lpNumFrames	LPAT_UINT	Number of frames.
lpDuration	LPAT_UINT	Duration of multimedia file in milliseconds.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_UINT numFrames;      /* Total number of frames in file */
AT_UINT msDuration;     /* Duration of file in milliseconds */
nErrcount = IG_mult_duration_get(hMult, &numFrames, &msDuration);
```

Remarks:

You can pass NULL for one of these arguments if you only need the other one. You can use this function to determine how many frames are in a multimedia instance and how long it would play for if played at normal speed.

1.3.1.2.26.16 IG_mult_frame_duration_get

This function returns the duration of the given frame in 100 nanosecond units.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_frame_duration_get(
        HIGMULT hMult,
        AT_UINT frameNum,
        LPAT_UINT lpDuration
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
frameNum	AT_UINT	Frame number to get duration of (0 = first frame).
lpDuration	LPAT_UINT	Frame duration (in 100ns units).

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Retrieve duration of first frame */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_UINT duration;       /* Duration of frame in 100ns units */
nErrcount = IG_mult_frame_duration_get(hMult, 0, &duration);
```

Remarks:

This provides a reasonable level of accuracy for the short frame durations that are typical in video. To convert to milliseconds, divide by 10000.

Note that retrieving the durations of individual frames is essential for proper timing when the frame rate is variable, as is often the case with animated GIF files.

1.3.1.2.26.17 IG_mult_frame_image_get

This function retrieves the HIGEAR of the given frame's image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_frame_image_get(
        HIGMULT hMult,
        AT_UINT frameNum,
        LPHIGEAR lphIGear
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
frameNum	AT_UINT	Frame number to get image for (0 = first frame).
lphIGear	LPHIGEAR	Image handle.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Saves first frame as a 128-pixel-wide thumbnail */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
HIGEAR hIGear;          /* HIGEAR handle of image */
HIGEAR hIGearCopy;      /* HIGEAR handle of duplicate */
AT_DIMENSION w, h;      /* Width and height of image */
nErrcount = IG_mult_current_frame_reset(hMult);
nErrcount = IG_mult_current_frame_image_get(hMult, &hIGear);
nErrcount = IG_image_duplicate(hIGear, &hIGearCopy);
nErrcount = IG_image_dimensions_get(hIGearCopy, &w, &h, NULL);
nErrcount = IG_IP_resize(hIGearCopy, 128, (AT_DIMENSION) ((float) h / (float) w * 128),
IG_INTERPOLATION_BICUBIC);
nErrcount = IG_save_file(hIGearCopy, "thumb.jpg", IG_SAVE_JPG);
nErrcount = IG_image_delete(hIGearCopy);
```

Remarks:

This image is owned by ImageGear. You can perform read-only operations such as display or save to a file, but do not attempt to free or alter the image. If you need to alter the image (resize it, for example), use IG_image_duplicate() to make a copy of the image, work with the copy, and free it with IG_image_delete() when finished.

1.3.1.2.26.18 IG_mult_frame_info_get

This function retrieves a specific piece of information about the current frame.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_frame_info_get(
        HIGMULT hMult,
        AT_UINT frameNum,
        AT_MODE infoID,
        LPAT_VOID lpInfo,
        LPAT_UINT lpInfoSize
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
frameNum	AT_UINT	Frame number to get info for (0 = first frame).
infoID	AT_MODE	ID of info to get.
lpInfo	LPAT_VOID	Pointer to buffer in which to store info.
lpInfoSize	LPAT_UINT	Size of buffer pointed to by lpInfo.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

You can specify the information to retrieve using the infoID parameter, whose value is a member of enumIGMultInfo defined in accucnst.h.

1.3.1.2.26.19 IG_mult_frame_info_set

This function sets a specific piece of information about the current frame.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_frame_info_set(
        HIGMULT hMult,
        AT_UINT frameNum,
        AT_MODE infoID,
        LPAT_VOID lpInfo,
        LPAT_UINT lpInfoSize
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
frameNum	AT_UINT	Frame number for which to set info.
infoID	AT_MODE	ID of info to set.
lpInfo	LPAT_VOID	Pointer to buffer containing info.
lpInfoSize	LPAT_UINT	Size of buffer pointed to by lpInfo.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Specify the information to retrieve using the infoID parameter, whose value is a member of enumIGMultInfo defined in accucnst.h.

1.3.1.2.26.20 IG_mult_frame_num_from_time_get

This function returns the number of the frame that would be visible at the given time under normal speed playback conditions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_frame_num_from_time_get(
        HIGMULT hMult,
        AT_UINT msTime,
        LPAT_UINT lpFrameNum
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
msTime	AT_UINT	Time in milliseconds (since the beginning).
lpFrameNum	LPAT_UINT	Frame number at given time.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Finds the # of the frame at 1 second from the beginning */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_UINT frameNum;       /* Returned frame number */
nErrcount = IG_mult_frame_num_from_time_get(hMult, 1000, &frameNum);
```

1.3.1.2.26.21 IG_mult_has_audio

This function fills a Boolean variable indicating whether or not a multimedia instance has audio data available.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_has_audio(
        HIGMULT hMult,
        LPAT_BOOL lpbHasAudio
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
lpbHasAudio	AT_BOOL	Pointer to Boolean set to TRUE if multimedia instance has audio.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;   /* Number of errors on stack */
HIGMULT hMult;           /* Multimedia instance handle */
AT_BOOL bHasAudio;       /* Is audio data available? */
nErrcount = IG_mult_has_audio(hMult, &bHasAudio);
```

1.3.1.2.26.22 IG_mult_info_get

This function retrieves a piece of information for a multimedia instance as a whole.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_info_get(
        HIGMULT hMult,
        AT_MODE infoID,
        LPAT_VOID lpInfo,
        LPAT_UINT lpInfoSize
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
infoID	AT_MODE	ID of info to get.
lpInfo	LPAT_VOID	Pointer to buffer in which to store info.
lpInfoSize	LPAT_UINT	Size of buffer pointed to by lpInfo.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Retrieve minimum frame delay setting for animated GIF */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_UINT msDelay;        /* Minimum frame delay (milliseconds) */
nErrcount = IG_mult_info_get(hMult, IG_MULT_INFO_GIF_MIN_DELAY, &msDelay,
sizeof(msDelay));
```

Remarks:

You can specify the information to retrieve using the infoID parameter, whose value is a member of enumIGMultInfo defined in accucnst.h.

1.3.1.2.26.23 IG_mult_has_video

This function fills a Boolean variable indicating whether or not a multimedia instance has video data available.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_has_video(
        HIGMULT hMult,
        LPAT_BOOL lpbHasVideo
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
lpbHasVideo	LPAT_BOOL	Pointer to Boolean set to TRUE if multimedia instance has video.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_BOOL bHasVideo;      /* Is video data available? */
nErrcount = IG_mult_has_video(hMult, &bHasVideo);
```

1.3.1.2.26.24 IG_mult_info_set

This function sets a piece of information for a multimedia instance as a whole.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_info_set(
        HIGMULT hMult,
        AT_MODE infoID,
        LPAT_VOID lpInfo,
        LPAT_UINT lpInfoSize
);
```

Arguments:

Name	Type	Description
hMult	HIGMULT	Multimedia instance handle.
infoID	AT_MODE	ID of info to set.
lpInfo	LPAT_VOID	Pointer to buffer containing info.
lpInfoSize	LPAT_UINT	Size of buffer pointed to by lpInfo.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Set minimum frame delay setting for animated GIF to 100ms */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
AT_UINT msDelay = 100;  /* Minimum frame delay (milliseconds) */
nErrcount = IG_mult_info_set(hMult, IG_MULT_INFO_GIF_MIN_DELAY, &msDelay,
sizeof(msDelay));
```

Remarks:

You can specify the information to retrieve using the infoID parameter, whose value is a member of enumIGMultInfo defined in accucnst.h.

1.3.1.2.26.25 IG_mult_open_FD

This function creates a multimedia instance from a file descriptor.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_open_FD(
        AT_INT fd,
        LPHIGMULT lphMult
);
```

Arguments:

Name	Type	Description
fd	AT_INT	File descriptor.
lphMult	LPHIGMULT	Handle to multimedia instance.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Get a multimedia file's duration using a file descriptor */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
LPCSTR szFile;          /* Name of file to open */
HANDLE fd;              /* File descriptor */
AT_UINT numFrames;      /* Total number of frames in file */
AT_UINT duration;       /* Duration of file in milliseconds */
fd = CreateFile(szFile, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
nErrcount = IG_mult_open_FD(fd, &hMult);
nErrcount = IG_mult_duration_get(hMult, &numFrames, &duration);
nErrcount = IG_mult_close(hMult);
CloseHandle(fd);
```

Remarks:

The file descriptor and contents must remain accessible until the multimedia instance is closed.

1.3.1.2.26.26 IG_mult_open_FD_format

This function creates a multimedia instance from a file descriptor using only the specified format.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_open_FD_format(
        AT_INT fd,
        AT_MODE format,
        LPHIGMULT lphMult
);
```

Arguments:

Name	Type	Description
fd	AT_INT	File descriptor.
format	AT_MODE	Format of file (can be IG_FORMAT_UNKNOWN).
lphMult	LPHIGMULT	Handle to multimedia instance.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Open an animated GIF file from a file descriptor */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
LPCSTR szFile;          /* Name of file to open */
HANDLE fd;              /* File descriptor */
AT_UINT numFrames;      /* Total number of frames in file */
AT_UINT duration;       /* Duration of file in milliseconds */
fd = CreateFile(szFile, GENERIC_READ, FILE_SHARE_READ, NULL, OPEN_ALWAYS,
FILE_ATTRIBUTE_NORMAL, NULL);
nErrcount = IG_mult_open_FD_format(fd, IG_FORMAT_GIF, &hMult);
nErrcount = IG_mult_duration_get(hMult, &numFrames, &duration);
nErrcount = IG_mult_close(hMult);
CloseHandle(fd);
```

Remarks:

The file descriptor and contents must remain accessible until the multimedia instance is closed. Specifying the format lets you bypass the automatic file type identification for situations in which you know the format of the file.

See the multimedia API overview for a list of currently supported formats.

1.3.1.2.26.27 IG_mult_open_file

This function creates a multimedia instance from a filename.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_open_file(
        LPSTR szFilename,
        LPHIGMULT lphMult
);
```

Arguments:

Name	Type	Description
szFilename	LPSTR	Filename of multimedia to open.
lphMult	LPHIGMULT	Handle to multimedia instance.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Get a multimedia file's duration using a filename */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
LPCSTR szFile;          /* Name of file to open */
AT_UINT numFrames;      /* Total number of frames in file */
AT_UINT duration;       /* Duration of file in milliseconds */
nErrcount = IG_mult_open_file(szFile, &hMult);
nErrcount = IG_mult_duration_get(hMult, &numFrames, &duration);
nErrcount = IG_mult_close(hMult);
```

Remarks:

The file must remain accessible until the multimedia instance is closed.

1.3.1.2.26.28 IG_mult_open_file_format

This function creates a multimedia instance from a filename using only the specified format.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_open_file_format(
        LPSTR szFilename,
        AT_MODE format,
        LPHIGMULT lphMult
);
```

Arguments:

Name	Type	Description
szFilename	LPSTR	Filename of multimedia to open.
format	AT_MODE	Format of file (can be IG_FORMAT_UNKNOWN).
lphMult	LPHIGMULT	Handle to multimedia instance.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Get an animated GIF file's duration using a filename */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
LPCSTR szFile;          /* Name of file to open */
AT_UINT numFrames;      /* Total number of frames in file */
AT_UINT duration;       /* Duration of file in milliseconds */
nErrcount = IG_mult_open_file_format(szFile, IG_FORMAT_GIF, &hMult);
nErrcount = IG_mult_duration_get(hMult, &numFrames, &duration);
nErrcount = IG_mult_close(hMult);
```

Remarks:

The file must remain accessible until the multimedia instance is closed. Specifying the format lets you bypass the automatic file type identification for situations in which you know the format of the file.

See the multimedia API overview for a list of currently supported formats.

1.3.1.2.26.29 IG_mult_open_mem

This function creates a multimedia instance from memory.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_open_mem(
        LPAT_VOID lpMem,
        AT_UINT memLen,
        LPHIGMULT lphMult
);
```

Arguments:

Name	Type	Description
lpMem	LPAT_VOID	Pointer to data in memory to open.
memLen	AT_UINT	Length of data in memory in bytes.
lphMult	LPHIGMULT	Handle to multimedia instance.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Get a multimedia file's duration using a memory buffer */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
LPAT_VOID lpMem;        /* Memory buffer with multimedia data */
AT_UINT memLen;         /* Length of data in memory buffer */
AT_UINT numFrames;      /* Total number of frames in file */
AT_UINT duration;       /* Duration of file in milliseconds */
nErrcount = IG_mult_open_mem(lpMem, memLen, &hMult);
nErrcount = IG_mult_duration_get(hMult, &numFrames, &duration);
nErrcount = IG_mult_close(hMult);
```

Remarks:

The memory must contain a complete multimedia file and remain accessible until the multimedia instance is closed.

1.3.1.2.26.30 IG_mult_open_mem_format

This function creates a multimedia instance from memory using only the specified format.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_mult_open_mem_format(
        LPAT_VOID lpMem,
        AT_UINT memLen,
        AT_MODE format,
        LPHIGMULT lphMult
);
```

Arguments:

Name	Type	Description
lpMem	LPAT_VOID	Pointer to data in memory to open.
memLen	AT_UINT	Length of data in memory in bytes.
format	AT_MODE	Format of file (can be IG_FORMAT_UNKNOWN).
lphMult	LPHIGMULT	Handle to multimedia instance.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/* Get an animated GIF file's duration using a memory buffer */
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGMULT hMult;          /* Multimedia instance handle */
LPAT_VOID lpMem;        /* Memory buffer with multimedia data */
AT_UINT memLen;         /* Length of data in memory buffer */
AT_UINT numFrames;      /* Total number of frames in file */
AT_UINT duration;       /* Duration of file in milliseconds */
nErrcount = IG_mult_open_mem_format(lpMem, memLen, IG_FORMAT_GIF, &hMult);
nErrcount = IG_mult_duration_get(hMult, &numFrames, &duration);
nErrcount = IG_mult_close(hMult);
```

Remarks:

The memory must contain a complete multimedia file and remain accessible until the multimedia instance is closed. Specifying the format lets you bypass the automatic file type identification for situations in which you know the format of the file.

See the multimedia API overview for a list of currently supported formats.

1.3.1.2.27 Palette Functions

This section provides information about the Palette group of functions.

- IG_palette_entry_get
- IG_palette_entry_set
- IG_palette_get
- IG_palette_load
- IG_palette_save
- IG_palette_set

1.3.1.2.27.1 IG_palette_entry_get

This function obtains from image hIGear's DIB palette, the palette entry indicated by nIndex.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_entry_get (
        HIGEAR hIGear,
        LPAT_RGB lpRGBEntry,
        UINT nIndex
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRGBEntry	LPAT_RGB	Far pointer to an AT_RGB struct to receive the three color values of the palette entry. (Note that this is not an AT_RGBQUAD struct. Also note that the order of the bytes is Blue, Green, Red in an AT_RGB struct.)
nIndex	UINT	Which palette entry to get, 0 to 255.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB - 1…8 bpp.

Example:

```
HIGEAR hIGear;            /* HIGEAR handle of image    */
AT_RGB rgbPaletteColor;   /* Will hold returned color   */
AT_ERRCOUNT nErrcount;         /* Returned count of errors on stack*/
/* Get palette entry 255: */
nErrcount = IG_palette_entry_get ( hIGear, rgbPaletteColor, 255 );
```

1.3.1.2.27.2 IG_palette_entry_set

This function sets a single palette entry in image hIGear's DIB palette.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_entry_set (
        HIGEAR hIGear,
        const LPAT_RGB lpRGBEntry,
        UINT nIndex
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpRGBEntry	const LPAT_RGB	Far pointer to an AT_RGB struct containing the three color values to be set into the palette entry. Note that this is not an AT_RGBQUAD struct. Also note that the order is Blue, Green, Red in an AT_RGB struct.
nIndex	UINT	Which palette entry to set, 0 to 255.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB - 1…8 bpp.

Example:

```
HIGEAR hIGear;              /* HIGEAR handle of image  */
AT_RGB rgbPaletteColor;     /* Will hold returned color */
AT_ERRCOUNT      Errcount;         /* Returned count of errors onstack */
/* Set palette entry 255 to a medium-bright yellow, intensity 175:  */
rgbPaletteColor.b = 0;     /* There's no blue in yellow */
rgbPaletteColor.r  =  rgbPaletteColor.g  =  175;
nErrcount = IG_palette_entry_set ( hIGear, &rgbPaletteColor, 255 );
```

1.3.1.2.27.3 IG_palette_get

This function obtains the referenced image's DIB palette, storing it in your array of AT_RGBQUAD structs pointed to by lpPalette.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_get (
        HIGEAR hIGear,
        LPAT_RGBQUAD lpPalette
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image.
lpPalette	LPAT_RGBQUAD	Far pointer to the first of an array of AT_RGBQUAD structs sufficient to hold the entire palette from the image's DIB.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB - 1…8 bpp.

Example:

```
HIGEAR hIGear;          /* HIGEAR handle of image     */
AT_ERRCOUNT nErrcount;      /* Returned count of errors on stack  */
AT_RGBQUAD rgbqPalette[256];  /* Array of AT_RGBQUAD structs    */
nErrcount = IG_palette_get ( hIGear, rgbqPalette );
```

Remarks:

The array must be large enough to hold the palette. For example, if an 8-bit image, lpPalette must point to the start of an array of 256 AT_RGBQUAD structs, therefore to an array of at least 256 x 4 = 1024 bytes. If the image is 24-bit, no error is set but no palette is returned.

See also function IG_palette_set() function.

To obtain an image's logical palette (the palette after mapping through the display LUTs, or for a 24-bit image the palette that would be used for displaying to an 8-bit device), use function IG_dspl_palette_create().

1.3.1.2.27.4 IG_palette_load

This function loads a palette that was saved using IG_palette_save() function.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_load (
        const LPSTR lpszFileName,
        LPAT_RGBQUAD lpPalette,
        LPUINT lpNumEntries,
        BOOL bBGR_Order,
        LPAT_MODE lpFileType
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Name of file containing saved palette to load.
lpPalette	LPAT_RGBQUAD	Far pointer to array of AT_RGBQUAD structs to load into (see the parameter bBGR_Order, below).
lpNumEntries	LPUINT	Far pointer to UINT variable to receive number of palette entries loaded (size of palette).
bBGR_Order	BOOL	TRUE = store as AT_RGB structs (that is, Blue-Green-Red), instead of AT_RGBQUAD structs (Blue-Green-Red-Unused). This switch has no effect when reading ImageGear's text format. It loads this information as if writing to AT_RGBQUAD.
lpFileType	LPAT_MODE	An IG_PALETTE_FORMAT_ constant specifying the format of the file. The constants are listed in file accucnst.h.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_RGBQUAD rgbqPalette[256];     /* Will hold the palette loaded */
UINT nEntries;             /* Holds number of entries in palette*/
AT_MODE nPaletteFileType;   /* Will receive IG_PALETTE_... constant*/
AT_ERRCOUNT1 nErrcount;            /* Returned count of errors    */
nErrcount = IG_palette_load ( "Palfile.pal", &rgbqPalette[0], &nEntries, TRUE,
&nPaletteFileType );
```

1.3.1.2.27.5 IG_palette_save

This function saves a palette to a file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_save (
        const LPSTR lpszFileName,
        LPAT_RGBQUAD lpPalette,
        UINT nNumEntries,
        AT_MODE nFileType
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	Name of file to which to save palette.
lpPalette	LPAT_RGBQUAD	Far pointer to array of AT_RGBQUAD structs constituting palette to save.
nNumEntries	UINT	Number of entries in palette.
nFileType	AT_MODE	IG_PALETTE_FORMAT_ constant specifying format in which to save palette. The constants are listed in accucnst.h.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR hIGear;      /* Handle of image whose palette to save  */
LPAT_RGBQUAD lpPalettePntr; /* Will hold address of the DIB palette  */
UINT nEntries;    /* Will hold number of entries in palette  */
/* Obtain address of DIB palette and its number of entries );
IG_image_DIB_palette_pntr_get ( hIGear, &lpPalettePntr, &nEntries );
/* Save to a file, saving only 3 bytes per entry, in order B-G-R: */
nErrcount = IG_palette_save ( "Savedpal.pal", lpPalettePntr, nEntries,
IG_PALETTE_FORMAT_RAW_BGR );
```

Remarks:

Argument nFileType lets you select the format in which to save the palette. If you save the palette using IG_PALETTE_FORMAT_TEXT, its format will be as shown for the following palette of a 1-bit black-and-white image:

```
Accusoft Palette File ver 7.0.9
0 0 0 0
1 255 255 255
```

The first line identifies the version of ImageGear that created this Palette text file. Then each succeeding line gives the entry number, followed by the Red, Green, and Blue color intensities, respectively, for that palette color.

1.3.1.2.27.6 IG_palette_set

This function loads the palette pointed to by lpPalette into the DIB, replacing the prior palette that was present.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_palette_set (
        HIGEAR hIGear,
        const LPAT_RGBQUAD lpPalette
);
```

Arguments:

Name	Type	Description
HIGEAR	HIGEAR	HIGEAR handle of image.
LpPalette	const LPAT_RGBQUAD	Far pointer to the first of an array of AT_RGBQUAD structs containing the palette you wish to load into the DIB, as the image's new DIB palette.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

Indexed RGB - 1…8 bpp.

Example:

```
HIGEAR hIGear;          /* HIGEAR handle of image  */
AT_ERRCOUNT  nErrcount;      /* Returned count of errors on stack */
AT_RGBQUAD rgbqPalette[256];   /* Array of AT_RGBQUAD structs   */
INT pix;                /* Loop index, = pixel value  */
/* Create a grayscale palette, and set it into image hIGear's DIB:  */
for ( pix = 0;  pix <= 255;  pix++ )
        {
        rgbqPalette[pix].rgbBlue  =  pix;
        rgbqPalette[pix].rgbGreen =  pix;
        rgbqPalette[pix].rgbRed   =  pix;
        rgbqPalette[pix].rgbReserved  =  0;
        }
nErrcount = IG_palette_set ( hIGear, rgbqPalette );
```

Remarks:

Your palette pointed to by lpPalette must be in the form of AT_RGBQUAD structs: 4 bytes per entry, ordered Blue-Green-Red-Unused (0). The number of consecutive AT_RGBQUAD structs you need is determined by the number of Bits Per Pixel in the image. For example, for an 8 bit image, you would need an array of 256 AT_RGBQUAD structs.

If the image is 24 bit, this function will set an error and return.

See also function IG_palette_set().

1.3.1.2.28 Pixel Functions

This section provides information about the Pixel group of functions.

- IG_pixel_bits_per_channel_get
- IG_pixel_channel_count_get
- IG_pixel_create
- IG_pixel_data_pointer_get
- IG_pixel_delete
- IG_pixel_value_get
- IG_pixel_value_set

1.3.1.2.28.1 IG_pixel_bits_per_channel_get

This function returns the number of bits (8, 16, or 32) allocated for storing pixel data for a single channel in the given pixel object.

Declaration:

```
AT_DEPTH ACCUAPI IG_pixel_bits_per_channel_get(HIGPIXEL hPixel);
```

Arguments:

Name	Type	Description
hPixel	HIGPIXEL	Handle of pixel object.

Return Value:

Return value is the same for all channels and represents the maximal bit depth aligned to all bytes.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DEPTH bitsPerChan;   /* Number of bits per channel */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_DIB_pix_get(hImage, 0, 0, &hPix);
bitsPerChan = IG_pixel_bits_per_channel_get(hPix);
IG_pixel_delete(hPix);
IG_image_delete(hImage);
```

1.3.1.2.28.2 IG_pixel_channel_count_get

This function returns the number of channels that are available to store values in the given pixel object.

Declaration:

```
AT_INT ACCUAPI IG_pixel_channel_count_get(HIGPIXEL hPixel );
```

Arguments:

Name	Type	Description
hPixel	HIGPIXEL	Handle of pixel object.

Return Value:

Number of channels in pixel object.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DEPTH nChannels;     /* Number of channels */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_DIB_pix_get(hImage, 0, 0, &hPix);
nChannels = IG_pixel_channel_count_get(hPix);
IG_pixel_delete(hPix);
IG_image_delete(hImage);
```

1.3.1.2.28.3 IG_pixel_create

This function creates a new pixel object based on specified attributes.

Declaration:

```
HIGPIXEL ACCUAPI IG_pixel_create(
        AT_INT channelCount,
        AT_DEPTH bitsPerChannel
);
```

Arguments:

Name	Type	Description
channelCount	AT_INT	Number of channels to allocate.
bitsPerChannel	AT_DEPTH	Bits per channel to allocate.

Return Value:

Handle of new pixel object.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_INT ChannelCount;    /* Number of channels in image */
AT_INT BitsPerChannel;  /* Bits per channel in image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &ChannelCount);
nErrcount = IG_image_bits_per_channel_get(hImage, &BitsPerChannel);
hPix = IG_pixel_create(ChannelCount, BitsPerChannel);
/* Pixel is created using same attributes as image */
IG_pixel_delete(hPix);
IG_image_delete(hImage);
```

Remarks:

The number of bits per channel is the amount to allocate, and it must be 8, 16, or 32.

1.3.1.2.28.4 IG_pixel_data_pointer_get

This function returns a pointer to the pixel data stored for a pixel.

Declaration:

```
LPAT_VOID ACCUAPI IG_pixel_data_pointer_get(HIGPIXEL hPixel);
```

Arguments:

Name	Type	Description
hPixel	HIGPIXEL	Handle of pixel object.

Return Value:

N/A

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_INT ChannelCount;    /* Number of channels in image */
AT_INT BitsPerChannel;  /* Bits per channel in image */
LPAT_VOID lpPixData;    /* Pixel data */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &ChannelCount);
nErrcount = IG_image_bits_per_channel_get(hImage, &BitsPerChannel);
/* Get the first pixel of an image */
nErrcount = IG_DIB_pix_get(hImage, 0, 0, &hPix);
/* Set the bits in all of its channels to 1's */
lpPixData = IG_pixel_data_pointer_get(hPix);
memset(lpPixData, 255, ChannelCount * (BitsPerChannel / 8));
/* Write the modified pixel data back to the image */
IG_DIB_pix_set(hImage, 0, 0, hPix);
IG_pixel_delete(hPix);
IG_image_delete(hImage);
```

Remarks:

The number of accessible bytes can be calculated by multiplying the number of bits allocated per channel (8, 16, or 32) by the number of channels allocated, then dividing by 8.

1.3.1.2.28.5 IG_pixel_delete

This function releases the allocated resources for a pixel object.

Declaration:

```
AT_VOID ACCUAPI IG_pixel_delete(HIGPIXEL hPixel);
```

Arguments:

Name	Type	Description
hPixel	HIGPIXEL	Handle of pixel object.

Return Value:

N/A

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
    for (x = 0; x < w / 2; x++)
    {
        nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
        for (c = 0; c < nChannels; c++)
        {
            IG_image_channel_depth_get(hImage, c, &nDepth);
            nDepth = (1 << nDepth) - 1;
            inverted = nDepth - IG_pixel_value_get(hPix, c);
            IG_pixel_value_set(hPix, c, inverted);
        }
        nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
        IG_pixel_delete(hPix);
    }
nErrcount = IG_save_file(hImage, OUTPUT_FILENAME,
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

1.3.1.2.28.6 IG_pixel_value_get

This function returns the value of the requested channel.

Declaration:

```
AT_UINT ACCUAPI IG_pixel_value_get(
        HIGPIXEL hPixel,
        AT_INT channel
);
```

Arguments:

Name	Type	Description
hPixel	HIGPIXEL	Handle of pixel object.
channel	AT_INT	Channel index from which to get value.

Return Value:

Channel value.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
    for (x = 0; x < w / 2; x++)
    {
        nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
        for (c = 0; c < nChannels; c++)
        {
            IG_image_channel_depth_get(hImage, c, &nDepth);
            nDepth = (1 << nDepth) - 1;
            inverted = nDepth - IG_pixel_value_get(hPix, c);
            IG_pixel_value_set(hPix, c, inverted);
        }
        nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
        IG_pixel_delete(hPix);
    }
nErrcount = IG_save_file(hImage, OUTPUT_FILENAME,
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

Remarks:

The range of possible values depends on the bit depth of the channel.

1.3.1.2.28.7 IG_pixel_value_set

This function updates the value for the specified channel.

Declaration:

```
AT_VOID ACCUAPI IG_pixel_value_set(
        HIGPIXEL hPixel,
        AT_INT channel,
        AT_UINT value
);
```

Arguments:

Name	Type	Description
hPixel	HIGPIXEL	Handle of pixel object.
channel	AT_INT	Channel index for which to set value.
value	AT_UINT	Value to set (range depends on channel bit depth).

Return Value:

N/A

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;  /* Number of errors on stack */
HIGEAR hImage;          /* Handle of image */
HIGPIXEL hPix;          /* Handle of pixel */
AT_DIMENSION w, h;      /* Width and height of image */
AT_INT nChannels;       /* Number of channels in image */
AT_DIMENSION x, y;      /* Used to loop over image */
AT_INT c;               /* Used to loop over channels */
AT_INT nDepth;          /* Channel depth */
AT_UINT inverted;       /* Inverted channel value */
/* Invert colors in upper-left quadrant of image */
nErrcount = IG_load_file("test.jpg", &hImage);
nErrcount = IG_image_channel_count_get(hImage, &nChannels);
nErrcount = IG_image_dimensions_get(hImage, &w, &h, NULL);
for (y = 0; y < h / 2; y++)
    for (x = 0; x < w / 2; x++)
    {
        nErrcount = IG_DIB_pix_get(hImage, x, y, &hPix);
        for (c = 0; c < nChannels; c++)
        {
            IG_image_channel_depth_get(hImage, c, &nDepth);
            nDepth = (1 << nDepth) - 1;
            inverted = nDepth - IG_pixel_value_get(hPix, c);
            IG_pixel_value_set(hPix, c, inverted);
        }
        nErrcount = IG_DIB_pix_set(hImage, x, y, hPix);
        IG_pixel_delete(hPix);
    }
nErrcount = IG_save_file(hImage, OUTPUT_FILENAME,
    IG_SAVE_BMP_UNCOMP);
IG_image_delete(hImage);
```

Remarks:

The range of possible values depends on the bit depth of the channel.

1.3.1.2.29 Resolution Unit Conversion Functions

This section provides information about the Resolution Unit Conversion group of functions.

- IG_convert_DPI_to_PPM
- IG_convert_PPM_to_DPI

1.3.1.2.29.1 IG_convert_DPI_to_PPM

Converts Dots Per Inch (DPI) to Pels Per Meter (PPM).

Declaration:

```
LONG ACCUAPI IG_convert_DPI_to_PPM(
        LONG lDotsPerInch
);
```

Arguments:

Name	Type	Description
lDotsPerInch	LONG	A variable of type LONG, holding the DPI value that can be converted to PPM (pels per meter).

Return Value:

Returns a LONG indicating the pels per meter of an image.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
LONG      IPpm, IDpi;
...
IPpm=IG_convert_DPI_to_PPM(IPpi);
...
```

Remarks:

"Pels" is an abbreviated term for pixels. This function can be useful when you are converting a file that supports dots per inch to a DIB format, which supports pels per meter. The header structure of a DIB (the BIMAPINFOHEADER) contains two fields whose values are defined in "pels per meter": LONG biXPelsPerMeter and LONG biYPelsPerMeter.

1.3.1.2.29.2 IG_convert_PPM_to_DPI

Converts Pels Per Meter (PPM) to Dots Per Inch (DPI).

Declaration:

```
LONG ACCUAPI IG_convert_PPM_to_DPI(
        LONG lPelsPerMeter
);
```

Arguments:

Name	Type	Description
lPelsPerMeter	LONG	A variable of type LONG, holding the pels per meter (PPM) value that can be converted to dots per inch (DPI).

Return Value:

Returns a LONG indicating the DPI of an image.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
LONG      IPpm, IDpi;
...
IPpm=IG_convert_PPM_to_DPI(IPpm);
...
```

Remarks:

"Pels" is an abbreviated term for pixels. This function can be useful when you are converting a DIB to a format that supports dots per inch. The header structure of a DIB (the BIMAPINFOHEADER) contains two fields whose values are defined in "pels per meter": LONG biXPelsPerMeter and LONG biYPelsPerMeter.

1.3.1.2.30 Run-End Functions

This section provides information about the Run-End group of functions.

- IG_runs_row_get
- IG_runs_row_set

1.3.1.2.30.1 IG_runs_row_get

This function returns a pointer to the run-end line specified by the yPos parameter.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_runs_row_get(
        HIGEAR hIGear,
        AT_PIXPOS yPos,
        AT_RUN* wRunCount,
        LPAT_RUN* lpRunEnd
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image from which to get the run ends information.
yPos	AT_PIXPOS	Y-Offset into the image.
wRunCount	AT_RUN*	The number of runs in the lpRunEnd buffer.
lpRunEnd	LPAT_RUN*	Returns a pointer to the run-end line specified by yPos.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB - 1 bpp;
Grayscale – 1bpp.

Example:

```
AT_ERRCOUNT      nErrcount;
HIGEAR  hIGear;
AT_PIXPOS       y;
WORD    wRunCount;
LPWORD  lpRunEnd, lpRunBuffer;
AT_DIMENSION    nWidth, nHeight;
IG_image_dimensions_get(hIGear, &nWidth, &nHeight, NULL);
/* calculate the maximum size of a raster line */
lpRunBuffer = (LPWORD)malloc((nWidth + 3) * sizeof(WORD));
/* invert the image */
for (y = 0; y < nHeight; y++)
{
        IG_runs_row_get(hIGear, y, &wRunCount, &lpRunEnd);
        if (lpRunEnd[0] != 0)
        {
                lpRunBuffer[0] = 0;
                for (wRunCount = 0; lpRunEnd[wRunCount] !=
                        nWidth; wRunCount++)
                        lpRunBuffer[wRunCount + 1] = lpRunEnd[wRunCount];
                wRunCount++;
                lpRunBuffer[wRunCount++] = (WORD)nWidth;
                lpRunBuffer[wRunCount++] = (WORD)nWidth;
                lpRunBuffer[wRunCount++] = (WORD)nWidth;
        }
        else
        {
                memcpy(lpRunBuffer, lpRunEnd + 1, (wRunCount - 1) * sizeof(WORD));
                lpRunBuffer[wRunCount - 1] = nWidth;
        }
        IG_runs_row_set(hIGear, y, wRunCount, lpRunBuffer);
}
InvalidateRect(hWnd, NULL, FALSE);
free(lpRunBuffer);
```

Remarks:

Run-end encoding is used on 1-bit images only. The wRunCount argument returns the number of runs to which lpRunEnd points. The read-only data in the pointer returned should not be changed because it may corrupt the image.

To safely change the data, use IG_runs_row_set(). Developers should be cautious when using the IG_runs_row_set() function because it is possible to corrupt the image by supplying invalid run-end data.

This function will set an error, if the image specified by hIGear is not in run-ends format or if the yPos parameter is greater than the height of the image.

The format of the run-end encoded data is as follows: Each line in the image starts with a value of type AT_RUN which stores the number of AT_RUN values used to hold the line. This value is equal to the number of runs in the line plus one (for the size value). The rest of the line consists of run ends of type AT_RUN. A run end specifies the first pixel position beyond the run of color. The run ends alternate between white and black, and start with white. The line ends with at least three run ends containing a value equal to the image's width. A 2500 pixel source line with black pixels in positions 0, 7, 23, and 30, would be encoded as runs: 10, 0, 1, 7, 8, 23, 31, 2500, 2500, 2500 (with 10 being the number of AT_RUN values used to store the encoded line.)

This function returns a pointer to the second AT_RUN value of the line in the lpRunEnd parameter, and returns the number of runs in wRunCount. The IG_runs_row_set() function copies the data in the lpRunEnd parameter to the second AT_RUN value of the line. The first AT_RUN value is set to the wRunCount parameter.

1.3.1.2.30.2 IG_runs_row_set

This function sets the run-end data of the line specified by the yPos parameter.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_runs_row_set(
        HIGEAR hIGear,
        AT_PIXPOS yPos,
        AT_RUN wRunCount,
        LPCAT_RUN lpRunEnd
);
```

Arguments:

hIGear	HIGEAR	HIGEAR handle of the image containing the raster line you would like to modify.
yPos	AT_PIXPOS	Y-Offset into the image.
wRunCount	AT_RUN	The number of runs in the lpRunEnd buffer.
lpRunEnd	LPCAT_RUN	The Run- end pointer to copy into the image.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB - 1 bpp;
Grayscale – 1bpp.

Example:

See the example for IG_runs_row_get().

Remarks:

The wRunCount parameter should be set to the number of runs contained in the lpRunEnd buffer. Run-end encoding is used on 1-bit images only. This function is faster than using IG_DIB_raster_set() for a 1-bit image. This function copies the data in the lpRunEnd parameter to the second AT_RUN value of the line. The first AT_RUN value is set to the wRunCount parameter.

Developers should be cautious when using this function because it is possible to corrupt the image by supplying invalid run-end data.

Both of these functions will set an error if the image specified by hIGear is not in run-ends format or if the yPos parameter is greater than the height of the image.

The format of the run-end encoded data is as follows: Each line in the image starts with a value of type AT_RUN which stores the number of AT_RUN values used to hold the line. This value is equal to the number of runs in the line plus one (for the size value). The rest of the line consists of run ends of type AT_RUN. A run end specifies the first pixel position beyond the run of color. The run ends alternate between white and black, and start with white. The line ends with at least three run ends containing a value equal to the image's width. A 2500 pixel source line with black pixels in positions 0, 7, 23, and 30, would be encoded as runs: 10, 0, 1, 7, 8, 23, 31, 2500, 2500, 2500 (with 10 being the number of AT_RUN values used to store the encoded line.)

See also IG_runs_row_get() function.

1.3.1.2.31 Save Functions

This section provides information about the Save group of functions.

- IG_save_FD
- IG_save_FD_CB
- IG_save_FD_CB_direct
- IG_save_FD_CB_ex
- IG_save_file
- IG_save_file_size_calc
- IG_save_JPEG_quality_get
- IG_save_JPEG_quality_set
- IG_save_mem
- IG_save_mem_CB
- IG_save_mem_CB_direct
- IG_save_mem_CB_ex
- IG_save_tag_CB_register
- IG_save_thumbnail_set

1.3.1.2.31.1 IG_save_FD

This function saves the image referenced by hIGear to a file that has already been opened, and for which your application has the File descriptor.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_FD(
   HIGEAR hIGear,
   AT_INT fd,
   UINT nPage,
   UINT nReserved,
   AT_LMODE lFormatType
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to save.
fd	AT_INT	Handle of the open file for saving the image. This handle can be obtained from Microsoft Windows function such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
nPage	UINT	If saving to a multi-page file, set this to the page number to insert this page as. Note that page numbers begin at 1, not 0. Otherwise set to 1.
nReserved	UINT	Reserved, should always be set = 0 for now.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
HIGEAR hIGear;             //ImageGear handle
HANDLE fd;                 //File Descriptor
AT_ERRCOUNT nErrcount;   //Number of errors on stack

// Load the image
nErrcount = IG_load_file("picture.tif", &hIGear);
if(nErrcount == 0)
{
    // Create a file for writing
    fd = CreateFile(_T("picture_new.tif"), GENERIC_WRITE,
        0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

    if(fd != INVALID_HANDLE_VALUE)
    {
        // Save the HIGEAR image as page 3 of file whose descriptor is fd:
        nErrcount = IG_save_FD(hIGear, (AT_INT)fd, 1, 0, IG_SAVE_TIF_UNCOMP);
        CloseHandle((HANDLE)fd);
    }
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

When saving to an existing file having a multi-page format, this function permits you to insert your image into the file at the page number you designate by argument nPage. If you want to append your image as the final page of the file, set nPage = IG_APPEND_PAGE. lFormatType should specify the format type and compression of the already existing file. If you do not know the format type you can first make a call to IG_info_get_FD_ex.

See Saving to a Disk File Using a File Descriptor Handle for additional information.

When saving to a non-multi-page format, this function will save a new single-image file of the format type and compression specified by lFormatType. Any previous version of the file will be lost. When saving to a non-multi-page format, set nPage = 1.

In order for an ImageGear append page operation to work properly, the file handle must point to the very beginning of the existing image, rather than to one of its pages, start of pixel data, or any custom wrapper preceding the image.

Appending and Inserting: While IG_APPEND_PAGE assures you that your loaded image will be appended to a pre-existing multi-page file, there are two other instances in which the value you assign to nPage will cause an append: if you set nPage to less than 1, or if you set nPage to greater than the number of pages in the file to which you are saving.

To summarize: ImageGear will insert your image to a pre-existing multi-image file if you set nPage to a value between 1 and the number of the last page in the file.

ImageGear supports the writing of tiled images for specific image formats, but does not support the insertion, replacement, or appending of individual tiles.

1.3.1.2.31.2 IG_save_FD_CB

This function saves the image to a file using user-defined callback functions. This is an obsolete function, see Remarks.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_FD_CB(
   AT_INT fd,
   UINT nPage,
   UINT nReserved,
   AT_LMODE lFormatType,
   LPFNIG_RASTER_GET lpfnRasterGet,
   LPFNIG_DIB_GET lpfnDIBGet,
   LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file for saving the image. This handle can be obtained from Microsoft Windows function such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
nPage	UINT	If saving to a multi-page file, set this to the page number to insert this page as. Note that page numbers begin at 1, not 0. Otherwise set to 1.
nReserved	UINT	Reserved, should be set to 0 for now.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats.
lpfnRasterGet	LPFNIG_RASTER_GET	Pointer to a function of type LPFNIG_RASTER_GET, which will be called for each raster line of the image, before that line is saved.
lpfnDIBGet	LPFNIG_DIB_GET	Pointer to a function of type LPFNIG_DIB_GET, which will be called just prior to saving the DIB header.
lpPrivateData	LPVOID	Pointer to a private data area. This pointer will be passed to the above two callback functions each time they are called.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

Actual set of pixel formats supported by this function can be narrower, depending on the implementation of the user-defined callback functions.

Example:

```
AT_ERRCOUNT ACCUAPI MyDIBGet(
    LPVOID lpPrivate,    // Private data passed in
    LPAT_DIB lpDIB,      // DIB structure to return
    LPAT_RGBQUAD lpRGB    // DIB palette to be set
    )
{
    // Convert user DIB info into (*lpDIB) structure and copy a user palette to lpRGB
    return 0;
}

AT_ERRCOUNT ACCUAPI MyRasterGet(
    LPVOID           lpPrivate, // Private data passed in
    LPAT_PIXEL       lpRaster,  // Raster line to set
    AT_PIXPOS        row,       // Y position in the image
    DWORD            rasterSize // Size of the raster line
    )
{
    // Copy user pixel data to lpRaster in the appropriate format
    return 0;
}

void Example_IG_save_FD_CB()
{
    HANDLE  fd;                            // File Descriptor handle
    AT_ERRCOUNT     nErrcount;            // Count of returned errors on stack
    HIGEAR hIGear;                        //ImageGear handle
    nErrcount = IG_load_file("picture.tif", &hIGear);
    if(nErrcount == 0)
    {
        // Create a file for writing
        fd = CreateFile(_T("picture_new.tif"), GENERIC_WRITE,
            0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
        if(fd != INVALID_HANDLE_VALUE)
        {
            nErrcount = IG_save_FD_CB((AT_INT)fd, 1, 0, IG_SAVE_TIF_UNCOMP, MyRasterGet,
MyDIBGet, NULL);
            CloseHandle(fd);
        }
        // Destroy the image
        IG_image_delete(hIGear);
    }
}
```

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_save_FD_CB_ex instead.

First, your lpfnDIBGet() callback is called. This function supplies ImageGear with the image's width, height, Bits Per Pixel, and all DIB information in the form of a DIB header. If the image requires a palette, this callback function also supplies the palette.

ImageGear then writes a header out to file fd, in the lFormatType format. Next, lpfnRasterGet() is called once for each raster line. ImageGear gets the raster line from the callback function. Then, it compresses the raster line (according to lFormatType) and writes the line to fd. (Note that the calls for the raster lines are not necessarily in order.)

Refer to the descriptions for callback function types LPFNIG_DIB_GET and LPFNIG_RASTER_GET in this chapter.

In order for an ImageGear append page operation to work properly, the file handle must point to the very beginning of the existing image, rather than to one of its pages, start of pixel data, or any custom wrapper preceding the image.

Appending and Inserting: While IG_APPEND_PAGE assures you that your loaded image will be appended to a pre-existing multi-page file, there are two other instances in which the value you assign to nPage will cause an append: if you set nPage to less than 1, or if you set nPage to greater than the number of pages in the file that you are saving to.

To summarize: ImageGear will insert your image to a pre-existing multi-image file if you set nPage to a value between 1 and the number of the last page in the file.

ImageGear supports the writing of tiled images for specific image formats, but does not support the insertion, replacement, or appending of individual tiles.

1.3.1.2.31.3 IG_save_FD_CB_direct

This function has been deprecated and will be removed from the public API in a future release. Please use IG_save_FD_CB_ex instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_FD_CB_direct(
   AT_INT fd,
   UINT nPage,
   UINT nReserved,
   AT_LMODE lFormatType,
   LPFNIG_DIRECT_RASTER_GET lpfnRasterGet,
   LPFNIG_DIB_GET lpfnDIBGet,
   LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Not used.
nPage	UINT	Not used.
nReserved	UINT	Not used.
lFormatType	AT_LMODE	Not used.
lpfnRasterGet	LPFNIG_DIRECT_RASTER_GET	Not used.
lpfnDIBGet	LPFNIG_DIB_GET	Not used.
lpPrivateData	LPVOID	Not used.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

N/A

1.3.1.2.31.4 IG_save_FD_CB_ex

This function saves the image to a file using user-defined callback functions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_FD_CB_ex(
   AT_INT fd,
   UINT nPage,
   UINT nReserved,
   AT_LMODE lFormatType,
   LPFNIG_RASTER_GET lpfnRasterGet,
   LPFNIG_DIB_GET_EX lpfnDIBGetEx,
   LPVOID lpPrivateData
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Handle of the open file for saving the image. This handle can be obtained from Microsoft Windows function such as CreateFile(), and cast to AT_INT for passing to the function parameter. FILE pointers returned by functions such as fopen(), and file handles returned by functions such as _sopen_s() are not supported.
nPage	UINT	If saving to a multi-page file, set this to the page number to insert this page as. Note that page numbers begin at 1, not 0. Otherwise set to 1.
nReserved	UINT	Reserved, should be set to 0 for now.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats.
lpfnRasterGet	LPFNIG_RASTER_GET	Pointer to a function of type LPFNIG_RASTER_GET, which will be called for each raster line of the image, before that line is saved.
lpfnDIBGetEx	LPFNIG_DIB_GET	Pointer to a function of type LPFNIG_DIB_GET, which will be called just prior to saving the DIB header.
lpPrivateData	LPVOID	Pointer to a private data area. This pointer will be passed to the above two callback functions each time they are called.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Actual set of pixel formats supported by this function can be narrower, depending on the implementation of the user-defined callback functions.

Example:

```
AT_ERRCOUNT ACCUAPI MyDIBGetEx(
    LPVOID lpPrivate,    // Private data passed in
    HIGDIBINFO* lphDIB  // DIB info object to return
    )
{
    // Convert user DIB info into lphDIB
    return 0;
}
```

```
AT_ERRCOUNT ACCUAPI MyRasterGetEx(
    LPVOID lpPrivate,    // Private data passed in
    LPAT_PIXEL lpRaster,// Raster line to set
    AT_PIXPOS row,      // Y position in the image
    DWORD rasterSize    // Size of the raster line
    )
{
    // Copy user pixel data to lpRaster in the appropriate format
    return 0;
}
void Example_IG_save_FD_CB_ex()
{
    HANDLE  fd;                            // File Descriptor handle
    AT_ERRCOUNT     nErrcount;            // Count of returned errors on stack
    HIGEAR hIGear;                        //ImageGear handle
    nErrcount = IG_load_file("picture.tif", &hIGear);
    if(nErrcount == 0)
    {
        // Create a file for writing
        fd = CreateFile(_T("picture_new.tif"), GENERIC_WRITE,
            0, NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

        if(fd != INVALID_HANDLE_VALUE)
        {
            nErrcount = IG_save_FD_CB_ex((AT_INT)fd, 1, 0, IG_SAVE_TIF_UNCOMP,
MyRasterGetEx, MyDIBGetEx, NULL);
            CloseHandle(fd);
        }
        // Destroy the image
        IG_image_delete(hIGear);
    }
}
```

Remarks:

First, your lpfnDIBGetEx() callback is called. This function supplies ImageGear with the image's width, height, bits per pixel, and all DIB information in the form of a HIGDIBINFO object.

ImageGear then writes a header out to file fd, in the lFormatType format. Next, lpfnRasterGet() is called once for each raster line. ImageGear gets the raster line from the callback function. Then, it compresses the raster line (according to lFormatType) and writes the line to fd. (Note that the calls for the raster lines are not necessarily in order.)

In order for an ImageGear append page operation to work properly, the file handle must point to the very beginning of the existing image, rather than to one of its pages, start of pixel data, or any custom wrapper preceding the image.

Appending and Inserting: While IG_APPEND_PAGE assures you that your loaded image will be appended to a pre-existing multi-page file, there are two other instances in which the value you assign to nPage will cause an append: if you set nPage to less than 1, or if you set nPage to greater than the number of pages in the file to which you are saving.

To summarize: ImageGear will insert your image to a pre-existing multi-image file if you set nPage to a value between 1 and the number of the last page in the file.

ImageGear supports the writing of tiled images for specific image formats, but does not support the insertion, replacement or appending of individual tiles.

1.3.1.2.31.5 IG_save_file

This function will store the image rectangle of the image referenced by hIGear to disk using the name lpszFilename.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_file(
   HIGEAR hIGear,
   const LPSTR lpszFileName,
   AT_LMODE lFormatType
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to save.
lpszFileName	const LPSTR	Pointer to the filename (you may include path with filename) in which to save.
lFormatType	AT_MODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Remarks:

lFormatType is used to set the format and compression (if applicable) of the output file. If you want to have ImageGear use the file extension provided in your filename string (lpszFilename) to determine the file format to save to, set lFormatType = IG_SAVE_UNKNOWN.

When an image is saved to a multi-page file format (for example, TIFF or DCX), if the file already exists then the new image is appended as a new page in the file. When an image is saved to a non-multi-page file format, if the file already exists it is simply overwritten; the previous version of the file is lost.

IG_fltr_save_file is an extended version of this function. It allows inserting or replacing pages in multi-page files.

Some file formats, such as TXT, JPEG, and others, may be saved with additional control, using IG_fltr_ctrl_get and IG_fltr_ctrl_set. See the description also in the section Using Format Filters API for Filter Control.

1.3.1.2.31.6 IG_save_file_size_calc

This function is used to determine the size that is required for saving the image to the file or memory buffer in the given format.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_file_size_calc(
   HIGEAR hIGear,
   AT_LMODE lFormatType,
   LPAT_UINT lpFileSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image on which to calculate the size.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats.
lpFileSize	LPAT_UINT	Returned argument showing the required size of the file or memory.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

This call may be used prior to calling IG_save_mem to determine the amount of memory that needs to be allocated.

1.3.1.2.31.7 IG_save_JPEG_quality_get

This function returns the current setting for JPEG quality.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_JPEG_quality_get(
   LPUINT lpQuality
);
```

Arguments:

Name	Type	Description
lpQuality	LPUINT	A pointer to a UINT variable which will receive the current setting for JPEG quality.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
UINT       lpQuality;
AT_ERRCOUNT nErrcount = IG_save_JPEG_quality_get(&lpQuality);
```

Remarks:

The quality level is the amount of data loss that will occur during JPEG compression. The default algorithm for JPEG compression used by ImageGear is a lossy scheme. This means that some data will always be lost during compression. Use this function to set the level of loss, where 100 means the least amount possible of pixel data will be lost during compression, and 1 allows the most loss (resulting in the smallest possible file after compression). Please see JPEG format filter description for more details.

JPEG quality is only used when an image is being saved, not during the decompression process.

This function has the same effect as using IG_fltr_ctrl_get to get the value of the "QUALITY" control parameter for IG_FORMAT_JPG format filter.

Use IG_save_JPEG_quality_set to change the current setting.

1.3.1.2.31.8 IG_save_JPEG_quality_set

This function sets the quality level for saving JPEG compressed images.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_JPEG_quality_set(
   UINT nQuality
);
```

Arguments:

Name	Type	Description
nQuality	UINT	An integer value from 1 to 100, where 100 represents the "highest quality" or least amount of pixel data lost.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount = IG_save_JPEG_quality_set(95);
```

Remarks:

The quality level is the amount of data loss that will occur during JPEG compression. The default algorithm for JPEG compression used by ImageGear is a lossy scheme. This means that some data will always be lost during compression. Use this function to set the level of loss, where 100 means the least amount possible of pixel data will be lost during compression, and 1 allows the most loss (resulting in the smallest possible file after compression). Please see JPEG format filter description for more details.

JPEG quality is only used when an image is being saved, not during the decompression process.

This function has the same effect as using IG_fltr_ctrl_set to set the "QUALITY" control parameter for IG_FORMAT_JPG format filter.

A setting of 100 does not give you "lossless" JPEG. If you wish to save as lossless, use IG_fltr_ctrl_set to set the "TYPE" control parameter for IG_FORMAT_JPG format filter to IG_JPG_LOSSLESS before saving.

Use IG_save_JPEG_quality_get to obtain the current setting.

1.3.1.2.31.9 IG_save_mem

This function saves the image referenced by hIGear in a memory block.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_mem(
   HIGEAR hIGear,
   LPVOID lpImage,
   AT_UINT nImageSize,
   AT_UINT nBufferSize,
   UINT nPage,
   UINT nReserved,
   AT_LMODE lFormatType,
   LPAT_UINT lpActualSize
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of image to save.
lpImage	LPVOID	Pointer to first byte of memory area in which to save.
nImageSize	AT_UINT	Size of the image if it already exists in the buffer, 0 otherwise.
nBufferSize	AT_UINT	Size of the memory buffer.
nPage	UINT	If saving to a multi-page file, set this to the page number to insert this page as. Note that page numbers begin at 1, not 0. Otherwise set to 1.
nReserved	UINT	Reserved, should always be set = 0 for now.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats.
lpActualSize	LPAT_UINT	Size of new file in memory will be returned in the variable pointed by this parameter. NULL is allowed.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT     nErrcount;             // Count of returned errors on stack
HIGEAR hIGear;                         // ImageGear handle
AT_BYTE* lpMemoryBlock;                 // Memory block to save the image to
AT_UINT nMaxSize;                     // Size of the memory block
nErrcount = IG_load_file("picture.bmp", &hIGear);

if(nErrcount == 0)
{
    // Get a required size of the memory block
    nErrcount = IG_save_file_size_calc ( hIGear, IG_SAVE_BMP_UNCOMP, &nMaxSize);
    // Allocate a memory block
    lpMemoryBlock = (AT_BYTE*)malloc(nMaxSize);
    // Save image to the memory block in BMP format without compression:
    nErrcount = IG_save_mem(hIGear, lpMemoryBlock, 0, nMaxSize, 1, 0, IG_SAVE_BMP_UNCOMP,
NULL);
```

```
    // Destroy the image
    IG_image_delete(hIGear);
    // Some usage of the image in the memory
    //...
    free(lpMemoryBlock);
}
```

Remarks:

You provide the total size of your memory area, nBufferSize, so ImageGear can avoid writing beyond the area you have reserved for the file image. The image file that results will be identical to what would have been written to disk had you used IG_save_FD: it will begin with a header, and will be in the format you have declared by argument lFormatType. After writing the entire new file to memory, the actual size of this in-memory file is returned to you in the AT_UINT variable pointed to by lpActualSize.

If the file format is multi-page, and if there already is a valid file of that format at location *lpImage, then the HIGEAR image you are saving will be inserted as the page number you've indicated by nPage. If you want to append your image to the multi-page file, set nPage = IG_APPEND_PAGE. If the file format is not multi-page, then any file image already at location lpImage will be overwritten. Set nPage = 1 for non-multi-page file formats.

It is your application's responsibility to allocate the memory to hold the file image, and to free this memory when it is no longer needed. You may call function IG_save_file_size_calc to determine the maximum amount of memory you need to allocate. (If you have not allocated enough memory an error will be set and *lpImage will contain an unfinished image. The image left in memory after this condition should not be used.)

In order for an ImageGear append page operation to work properly, the memory buffer must point to the very beginning of the existing image, rather than to one of its pages, start of pixel data, or any custom wrapper preceding the image.

Appending and Inserting: While IG_APPEND_PAGE assures you that your loaded image will be appended to a pre-existing multi-page file, there are two other instances in which the value you assign to nPage will cause an append: if you set nPage to less than 1, or if you set nPage to greater than the number of pages in the file to which you are saving.

To summarize: ImageGear will insert your image to a pre-existing multi-image file if you set nPage to a value between 1 and the number of the last page in the file.

ImageGear supports the writing of tiled images for specific image formats, but does not support the insertion, replacement or appending of individual tiles.

1.3.1.2.31.10 IG_save_mem_CB

This function saves the image referenced by hIGear to a memory buffer using user-defined callback functions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_mem_CB(
   LPVOID lpImage,
   AT_UINT nImageSize,
   AT_UINT nBufferSize,
   UINT nPage,
   UINT nReserved,
   AT_LMODE lFormatType,
   LPFNIG_RASTER_GET lpfnRasterGet,
   LPFNIG_DIB_GET lpfnDIBGet,
   LPVOID lpPrivateData,
   LPAT_UINT lpActualSize
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Memory buffer to which to save the image.
nImageSize	AT_UINT	Size of the image if it already exists in the buffer, 0 otherwise.
nBufferSize	AT_UINT	Size of the memory buffer.
nPage	UINT	If saving to a multi-page file, set this to the page number to insert this page as. Note that page numbers begin at 1, not 0. Otherwise set to 1.
nReserved	UINT	Reserved, should be set to 0 for now.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats.
lpfnRasterGet	LPFNIG_RASTER_GET	Pointer to a function of type LPFNIG_RASTER_GET, which will be called for each raster line of the image, before that line is saved.
lpfnDIBGet	LPFNIG_DIB_GET	Pointer to a function of type LPFNIG_DIB_GET, which will be called just prior to saving the DIB header.
lpPrivateData	LPVOID	Pointer to a private data area which is passed to the above two callback functions each time they are called.
lpActualSize	LPAT_UINT	Actual size of the image is returned in the variable referenced by this pointer. Can be NULL.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

- Indexed RGB - 1, 4, 8 bpp;
- Grayscale - 9...16 bpp;
- RGB - 24 bpp;
- CMYK - 32 bpp.

Actual set of pixel formats supported by this function can be narrower, depending on the implementation of the user-defined callback functions.

Remarks:

This function is only kept for backward compatibility reasons. Please use IG_save_mem_CB_ex instead.

1.3.1.2.31.11 IG_save_mem_CB_direct

This function has been deprecated and will be removed from the public API in a future release. Please use IG_save_mem_CB_ex instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_mem_CB_direct(
   LPVOID lpImage,
   AT_UINT nImageSize,
   AT_UINT nBufferSize,
   UINT nPage,
   UINT nReserved,
   AT_LMODE lFormatType,
   LPFNIG_DIRECT_RASTER_GET lpfnRasterGet,
   LPFNIG_DIB_GET lpfnDIBGet,
   LPVOID lpPrivateData,
   LPAT_UINT lpActualSize
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Not used.
nImageSize	AT_UINT	Not used.
nBufferSize	AT_UINT	Not used.
nPage	UINT	Not used.
nReserved	UINT	Not used.
lFormatType	AT_LMODE	Not used.
lpfnRasterGet	LPFNIG_DIRECT_RASTER_GET	Not used.
lpfnDIBGet	LPFNIG_DIB_GET	Not used.
lpPrivateData	LPVOID	Not used.
lpActualSize	LPAT_UINT	Not used.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

N/A

1.3.1.2.31.12 IG_save_mem_CB_ex

This function saves the image referenced by hIGear to a memory buffer using user-defined callback functions.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_mem_CB_ex(
   LPVOID lpImage,
   AT_UINT nImageSize,
   AT_UINT nBufferSize,
   UINT nPage,
   UINT nReserved,
   AT_LMODE lFormatType,
   LPFNIG_RASTER_GET lpfnRasterGet,
   LPFNIG_DIB_GET_EX lpfnDIBGetEx,
   LPVOID lpPrivateData,
   LPAT_UINT lpActualSize
);
```

Arguments:

Name	Type	Description
lpImage	LPVOID	Memory buffer to which to save the image.
nImageSize	AT_UINT	Size of the image if it already exists in the buffer, 0 otherwise.
nBufferSize	AT_UINT	Size of the memory buffer.
nPage	UINT	If saving to a multi-page file, set this to the page number to insert this page as. Note that page numbers begin at 1, not 0. Otherwise set to 1.
nReserved	UINT	Reserved, should be set to 0 for now.
lFormatType	AT_LMODE	Specifies the format to use for saving, and also the compression scheme if applicable. See enumIGSaveFormats.
lpfnRasterGet	LPFNIG_RASTER_GET	Pointer to a function of type LPFNIG_RASTER_GET, which will be called for each raster line of the image, before that line is saved.
lpfnDIBGetEx	LPFNIG_DIB_GET	Pointer to a function of type LPFNIG_DIB_GET, which will be called just prior to saving the DIB header.
lpPrivateData	LPVOID	Pointer to a private data area which is passed to the above two callback functions each time they are called.
lpActualSize	LPAT_UINT	Actual size of the image is returned in the variable referenced by this pointer. Can be NULL.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Actual set of pixel formats supported by this function can be narrower, depending on the implementation of the user-defined callback functions.

Example:

```
AT_ERRCOUNT     nErrcount;              // Count of returned errors on stack
HIGEAR hIGear;                          //ImageGear handle
AT_BYTE* lpMemoryBlock;                  // Memory block to save the image to
```

```
AT_UINT nMaxSize;                    // Size of the memory block

nErrcount = IG_load_file("picture.bmp", &hIGear);
if(nErrcount == 0)
{
    // Get a required size of the memory block
    nErrcount = IG_save_file_size_calc ( hIGear, IG_SAVE_BMP_UNCOMP, &nMaxSize);
    // Allocate a memory block
    lpMemoryBlock = (AT_BYTE*)malloc(nMaxSize);
    // Save image to the memory block in BMP format without compression:
    nErrcount = IG_save_mem_CB_ex(lpMemoryBlock, 0, nMaxSize, 1, 0, IG_SAVE_BMP_UNCOMP,
        MyRasterGetEx, MyDIBGetEx, &hIGear, NULL);
    // Destroy the image
    IG_image_delete(hIGear);
    // Some usage of the image in the memory
    //...
    free(lpMemoryBlock);
}
```

Remarks:

This function works similarly to IG_save_FD_CB_ex, except that the saving is made to a memory buffer rather than a file.

In order for an ImageGear append page operation to work properly, the memory buffer must point to the very beginning of the existing image, rather than to one of its pages, start of pixel data, or any custom wrapper preceding the image.

1.3.1.2.31.13 IG_save_tag_CB_register

This function has been deprecated and will be removed from the public API in a future release. Please use IG_fltr_metad_callback_set instead.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_tag_CB_register(
   LPFNIG_TAG_GET lpfnTagGet,
   LPFNIG_TAG_USER_GET lpfnTagUserGet,
   LPVOID lpPrivate
);
```

Arguments:

Name	Type	Description
lpfnTagGet	LPFNIG_TAG_GET	Pointer to callback function to be called during save operationS, prior to saving each Tag.
lpfnTagUserGet	LPFNIG_TAG_USER_GET	Pointer to callback function to be called to obtain additional user Tags.
lpPrivate	LPVOID	Pointer to private data (passed to callback function).

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
LPFNIG_TAG_GET    MyTagChanger = NULL;        // To change existing tags
LPFNIG_TAG_USER_GET  MyTiffTagger = NULL;    // Can add new tags
char* lpPrivate = NULL;                        // Pointer to private area
IG_save_tag_CB_register ( MyTagChanger, MyTiffTagger, (LPVOID) lpPrivate );
```

Remarks:

This function registers a callback function to be called for each Tag while saving a file.

A default value of the Tag is supplied, and can be changed in the callback function. See the description for callback function type LPFNIG_TAG_GET.

The second callback function you supply is of type LPFNIG_TAG_USER_GET, and permits you to provide additional tags. This can be used to add additional TIFF Tags when saving in TIFF format.

1.3.1.2.31.14 IG_save_thumbnail_set

This function allows you to save a thumbnail (miniature) version of the image together with the full image, if the format that you are saving to supports thumbnails.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_save_thumbnail_set(
   AT_BOOL bSaveThumbnails,
   AT_DIMENSION nWidth,
   AT_DIMENSION nHeight
);
```

Arguments:

Name	Type	Description
bSaveThumbnails	AT_BOOL	Thumbnail Flag: TRUE = enable saving thumbnails; FALSE = disable saving thumbnails.
nWidth	AT_DIMENSION	Width of thumbnail rectangle.
nHeight	AT_DIMENSION	Height of thumbnail rectangle.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR hIGear;              // Will hold handle ret'd by IG_load_file  */
AT_ERRCOUNT nErrcount;      // Count of errs on stack upon ret from func */

nErrcount = IG_load_file("picture.tif", &hIGear);
if (nErrcount == 0)
{
    // To the file to be saved, add a thumbnail version of processed image
    IG_save_thumbnail_set(TRUE, 32, 32);
    nErrcount = IG_save_file (hIGear, "picture_new.jpg", IG_SAVE_JPG);
    // Destroy the image
    IG_image_delete(hIGear);
}
```

Remarks:

This function currently affects the following format filters: JPEG, PSB, PSD and Targa.

For Targa format, thumbnail cannot exceed 64 x 64 pixels. There is no limitation on thumbnail dimensions in JPEG-JFIF, but the result thumbnail image size should not exceed approximately 65536 bytes. Targa and JPEG-JFIF thumbnails are always 8-bit images. If the image data has a bit depth of greater than 8, ImageGear will automatically reduce the number of bits when creating the thumbnail.

When you load an image using one of the IG_load_...() functions, you will not automatically load any thumbnail that accompanies the main image. To load the thumbnail you must subsequently call one of the IG_load_thumbnail_...() functions. Similarly, when calling any of the IG_save_...() functions, the thumbnail will not be saved unless you first make a call to IG_save_thumbnail_set() function.

An alternative way to enable thumbnail saving is using filter control parameters. See descriptions of corresponding format filters in ImageGear Supported File Formats Reference.

If you load a file that has a thumbnail, and save it without having thumbnail saving enabled via IG_save_thumbnail_set() or via control parameters, only the main image will be saved to the destination file; no thumbnail will be saved.

1.3.1.2.32 Thread Functions

This section provides information about the Thread group of functions.

- IG_thread_data_ID_associate
- IG_thread_data_ID_get
- IG_thread_local_data_cleanup
- IG_thread_image_lock
- IG_thread_image_unlock

1.3.1.2.32.1 IG_thread_data_ID_associate

ImageGear allows you to have different groups of settings for providing mechanisms for associating a thread with a particular group.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_thread_data_ID_associate(
        DWORD dwNewId,
        AT_BOOL bLeaveThreadStorage
);
```

Arguments:

Name	Type	Description
dwNewId	DWORD	ID of a new Data storage.
bLeaveThreadStorage	AT_BOOL	If TRUE, old data storage is left in memory even if unused by any of the threads.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrCount = 0;
nErrCount = IG_comm_comp_attach("ART");
. . .
AfxBeginThread( IGProcThread1, NULL );
AfxBeginThread( IGProcThread2 , NULL );
UINT IGProcThread1( LPVOID lpData )
{
        AT_ERRCOUNT nErrCnt;
        nErrCnt = IG_thread_data_ID_associate( 1, TRUE );
        nErrCnt = IG_load_file( "picture1.bmp", &hIGear);
        . . .
        return 0;
}
UINT IGProcThread2( LPVOID lpData )
{
        AT_ERRCOUNT nErrCnt;
        nErrCnt = IG_thread_data_ID_associate( 2, TRUE );
        nErrCnt = IG_load_file( "picture2.bmp", &hIGear);
        . . .
        return 0;
}:
```

Remarks:

By default, each thread uses the global copy. Such a mechanism allows you to have ImageGear settings ("groups" that are identified by specific IDs) that are customized for each thread. This API allows you to associate a thread with different ImageGear settings. Allocation of new settings will result in the use of that group's default values if it doesn't exist prior to the call. The old copy of the settings will be deleted unless it's (a) a global copy used by other threads, or (b) bLeaveThreadStorage is TRUE.

The global group of ART settings cannot be used in multi-threaded application. You have to use this function to associate a new local group of settings for each new thread.

The access to the same PDF document from multiple threads is not permitted because the multiple threads cannot share Adobe PDF Library data types. PDF doc created/opened in the main thread can be only used from the main thread.

1.3.1.2.32.2 IG_thread_data_ID_get

This function retrieves the ID of the group of settings used by a particular thread.

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_thread_data_ID_get(LPDWORD lpdwDataId);
```

Arguments:

Name	Type	Description
LpdwDataId	LPDWORD	A pointer to a variable for receiving information about group ID of settings currently associated with a thread.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
nErrCnt = IG_thread_data_ID_get( &dwLocalId );
```

1.3.1.2.32.3 IG_thread_local_data_cleanup

This function destroys a group of settings with a specific ID if it's not in use by any threads.

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_thread_local_data_cleanup(DWORD dwClDataId);
```

Arguments:

Name	Type	Description
dwClDataId	DWORD	The group ID of settings to clean up.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
nErrCnt = IG_thread_local_data_cleanup( 1 );
```

1.3.1.2.32.4 IG_thread_image_lock

This function locks HIGEAR for a particular operation.

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_thread_image_lock(
        HIGEAR hIGear,
        AT_MODE nLockMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR image handle
nLockMode	AT_MODE	The lock mode (IG_THREAD_LOCK_READ or IG_THREAD_LOCK_WRITE) indicator. Several threads can perform read lock concurrently, but only one thread is allowed to do write lock.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
hIGear = pDoc->hMainGear;
        nErrCnt = IG_thread_image_lock( pDoc->hMainGear, IG_THREAD_LOCK_WRITE );
        if (nErrCnt == 0)
                {
                // locked successfully
                IG_IP_rotate_multiple_90( hIGear, IG_ROTATE_90 );
                IG_thread_image_unlock( pDoc->hMainGear, IG_THREAD_LOCK_WRITE );
...
                }
```

Remarks:

Two APIs (IG_thread_image_lock(), IG_thread_image_unlock()) are required for those rare situations in which two or more threads-at least one of which is modifying or deleting an image-are simultaneously accessing the same HIGEAR. For example, one thread can save HIGEAR to a JPEG file (read access to a HIGEAR), while another is performing rotation in the asynchronous mode (write access to a HIGEAR). Thus, the "saving" thread would be required to call lock/unlock with the read mode as a parameter, and the "rotation" thread would be required to call lock/unlock with the write mode as a parameter.

Please note that all auxiliary HIGEARs associated with the main HIGEAR (such as Alpha channel, NRA mask, and transparency mask) need to be locked separately.

1.3.1.2.32.5 IG_thread_image_unlock

This function unlocks the specified locked HIGEAR.

Declaration:

```
AT_ERRCOUNT EXPORT ACCUAPI IG_thread_image_unlock(
        HIGEAR hIGear,
        AT_MODE nLockMode
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	ImageGear Image handle.
nLockMode	AT_MODE	The lock mode (IG_THREAD_LOCK_READ or IG_THREAD_LOCK_WRITE) indicator. Several threads can perform read lock concurrently, but only one thread is allowed to do write lock.

Return Value:

Returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
hIGear = pDoc->hMainGear;
        nErrCnt = IG_thread_image_lock( pDoc->hMainGear, IG_THREAD_LOCK_WRITE );
        if (nErrCnt == 0)
                {
                // locked successfully
                IG_IP_rotate_multiple_90( hIGear, IG_ROTATE_90 );
                IG_thread_image_unlock( pDoc->hMainGear, IG_THREAD_LOCK_WRITE );
...
                }
```

Remarks:

Two APIs (IG_thread_image_lock() and IG_thread_image_unlock()) are required for those situations in which two or more threads-at least one of which is modifying or deleting an image-are simultaneously accessing the same HIGEAR. For example, one thread can save HIGEAR to a JPEG file (read access to a HIGEAR), while another is performing rotation in the asynchronous mode (write access to a HIGEAR). Thus the "saving" thread would be required to call lock/unlock with the read mode as a parameter, and the "rotation" thread would be required to call lock/unlock with the write mode as a parameter.

1.3.1.2.33 Utility Functions

This section provides information about the Utility group of functions.

- IG_util_colorspace_alpha_count_get
- IG_util_colorspace_color_count_get
- IG_util_colorspace_contains_alpha
- IG_util_colorspace_contains_extra
- IG_util_colorspace_extra_count_get
- IG_util_colorspace_is_premultiplied
- IG_util_colorspace_is_valid
- IG_util_colorspace_value_to_ids
- IG_util_MMX_usage_get
- IG_util_MMX_usage_set
- IG_util_resolution_units_convert
- IG_util_version_get

1.3.1.2.33.1 IG_util_colorspace_alpha_count_get

This function returns the number of alpha channels (0 or 1) in the specified color space.

Declaration:

```
AT_UINT ACCUAPI IG_util_colorspace_alpha_count_get(
        enumIGColorSpaceIDs colorSpace
);
```

Arguments:

Name	Type	Description
colorSpace	enumIGColorSpaceIDs	Color space ID.

Return Value:

The number of alpha channels.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBA;
AT_UINT nAlpha;         /* Number of alpha channels */
nAlpha = IG_util_colorspace_alpha_count_get(colorSpace);
/* nAlpha is 1 */
```

1.3.1.2.33.2 IG_util_colorspace_color_count_get

This function returns the number of color channels in the specified color space.

Declaration:

```
AT_UINT ACCUAPI IG_util_colorspace_color_count_get(
        enumIGColorSpaceIDs colorSpace
);
```

Arguments:

Name	Type	Description
colorSpace	enumIGColorSpaceIDs	Color space ID.

Return Value:

The number of color channels.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBA;
AT_UINT nColor;         /* Number of color channels */
nColor = IG_util_colorspace_color_count_get(colorSpace);
/* nColor is 3 */
```

Remarks:

Color channels are all image channels except alpha and extra channels.

1.3.1.2.33.3 IG_util_colorspace_contains_alpha

This function returns whether or not the color space contains an alpha channel (either pre-multiplied or not pre-multiplied).

Declaration:

```
AT_BOOL ACCUAPI IG_util_colorspace_contains_alpha(
        enumIGColorSpaceIDs colorSpace
);
```

Arguments:

Name	Type	Description
colorSpace	enumIGColorSpaceIDs	Color space ID.

Return Value:

Returns TRUE if the color space contains an alpha channel; FALSE otherwise.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBA;
AT_BOOL bAlpha;         /* Does the color space have alpha? */
bAlpha = IG_util_colorspace_contains_alpha(colorSpace);
/* bAlpha is TRUE */
```

1.3.1.2.33.4 IG_util_colorspace_contains_extra

This function returns whether or not the color space contains extra channels.

Declaration:

```
AT_BOOL ACCUAPI IG_util_colorspace_contains_extra(
        enumIGColorSpaceIDs colorSpace
);
```

Arguments:

Name	Type	Description
colorSpace	enumIGColorSpaceIDs	Color space ID.

Return Value:

Returns TRUE if the color space contains extra channels; FALSE otherwise.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBAEx;
AT_BOOL bExtra; /* Does the color space have extra channels? */
bExtra = IG_util_colorspace_contains_extra(colorSpace);
/* bExtra is TRUE */
```

1.3.1.2.33.5 IG_util_colorspace_extra_count_get

This function returns the number of extra channels in the specified color space.

Declaration:

```
AT_UINT ACCUAPI IG_util_colorspace_extra_count_get(
        enumIGColorSpaceIDs colorSpace,
        AT_UINT totalChannelCount
);
```

Arguments:

Name	Type	Description
colorSpace	enumIGColorSpaceIDs	Color space ID.
totalChannelCount	AT_UINT	Total number of channels in the color space.

Return Value:

Returns the number of extra channels.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBAEx;
AT_UINT nExtra;         /* Number of extra channels */
nExtra = IG_util_colorspace_extra_count_get(colorSpace);
/* nExtra is 1 */
```

Remarks:

You must specify the total number of channels in the color space in order for this to be calculated.

1.3.1.2.33.6 IG_util_colorspace_is_premultiplied

This function checks whether the alpha channel in the color space is pre-multiplied.

Declaration:

```
AT_BOOL ACCUAPI IG_util_colorspace_is_premultiplied(
        enumIGColorSpaceIDs colorSpace
);
```

Arguments:

colorSpace	enumIGColorSpaceIDs	Color space ID.

Return Value:

Returns TRUE, if the alpha channel in the color space is pre-multiplied. Returns FALSE, if the alpha channel in the color space is not pre-multiplied, or the color space does not contain an alpha channel.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumIGColorSpaceIDs colorSpace = IG_COLOR_SPACE_ID_RGBA;
AT_BOOL bPremult; /* Does color space have premult. alpha? */
bPremult = IG_util_colorspace_is_premultiplied(colorSpace);
/* bPremult is FALSE */
```

1.3.1.2.33.7 IG_util_colorspace_is_valid

This function checks whether ImageGear supports the color space identified by the color space constant.

Declaration:

```
AT_BOOL ACCUAPI IG_util_colorspace_is_valid(
        enumIGColorSpaceIDs colorSpace
);
```

Arguments:

Name	Type	Description
colorSpace	enumIGColorSpaceIDs	Color space ID.

Return Value:

Returns TRUE, if the color space is supported; FALSE otherwise.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumIGColorSpaceIDs colorSpace;
/* Invalid color space - can't have non-premult and premult */
colorSpace = IG_COLOR_SPACE_ID_RGBA | IG_COLOR_SPACE_ID_P;
AT_BOOL bValid;         /* Is the color space valid? */
bValid = IG_util_colorspace_is_valid(colorSpace);
/* bValid is FALSE */
```

1.3.1.2.33.8 IG_util_colorspace_value_to_ids

This function maps from legacy ImageGear color space ID (enumColorSpaces) to current ImageGear color space ID (enumIGColorSpaceIDs).

Declaration:

```
enumIGColorSpaceIDs ACCUAPI IG_util_colorspace_value_to_ids(
        enumColorSpaces colorSpace
);
```

Arguments:

Name	Type	Description
colorSpace	enumColorSpaces	Legacy ImageGear color space ID.

Return Value:

The current ImageGear color space ID that corresponds with the given legacy ImageGear color space ID.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
enumColorSpaces oldCS;  /* Old color space ID */
enumIGColorSpaceIDs cs; /* New color space ID */
oldCS = IG_COLOR_SPACE_CMYK;
cs = IG_util_colorspace_value_to_ids(oldCS);
/* cs is now IG_COLOR_SPACE_ID_CMYK */
oldCS = IG_COLOR_SPACE_RGBA;
cs = IG_util_colorspace_value_to_ids(oldCS);
/* cs is now IG_COLOR_SPACE_ID_RGBA */
```

Remarks:

This function exists because the newer color space IDs are different and can store more information. See accucnst.h for the definitions of these enumerations.

1.3.1.2.33.9 IG_util_MMX_usage_get

This function returns the current state of MMX optimization.

Declaration:

```
AT_ERRCOUNT LACCUAPI IG_util_MMX_usage_get (LPBOOL lpbMMXUsage);
```

Arguments:

Name	Type	Description
lpbMMXUsage	LPBOOL	A far pointer that returns a Boolean value indicating whether or not ImageGear is set to optimize for MMX hardware.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_ERRCOUNT nErrcount;
BOOL bMMXon;
nErrcount = IG_util_MMX_usage_get(bMMXon);
```

Remarks:

If lpbMMXUsage returns TRUE it means that ImageGear is set to optimize for MMX technology if the computer contains MMX hardware.

If this parameter is TRUE, but the MMX processor is not detected, then MMX optimization will not be used.

See also IG_util_MMX_usage_set() function.

1.3.1.2.33.10 IG_util_MMX_usage_set

This function tells ImageGear whether or not to optimize for MMX hardware.

Declaration:

```
AT_ERRCOUNT LACCUAPI IG_util_MMX_usage_set(BOOL bMMXUsage);
```

Arguments:

Name	Type	Description
bMMXUsage	BOOL	Set to TRUE to turn on MMX optimization; FALSE to turn off MMX optimization.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

The JPEG sample demonstrates loading images with or without MMX support.

Example:

```
AT_ERRCOUNT   nErrcount;
bMMXusage = TRUE;
nErrcount = IG_util_MMX_usage_set(bMMXusage);
```

Remarks:

Set lpbMMXUsage to TRUE to instruct ImageGear to optimize for MMX technology if the computer contains MMX hardware.

If you set bMMXUsage = TRUE, MMX technology will be used if it is detected. If it is not detected, no error will be generated, and MMX optimization will not be used.

Currently, the JPEG Lossy Compression is the target area for optimization.

See also IG_util_MMX_usage_get() function.

1.3.1.2.33.11 IG_util_resolution_units_convert

This utility converts resolution data into new units.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_util_resolution_units_convert(
        AT_RESOLUTION* Resolution,
        enumIGResolutionUnits NewUnits
);
```

Arguments:

Name	Type	Description
Resolution	AT_RESOLUTION*	Pointer to structure containing resolution info.
NewUnits	enumIGResolutionUnits	New resolution units.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
// Initialize resolution as 300 DPI
AT_RESOLUTION Resolution = {300, 1, 300, 1, IG_RESOLUTION_INCHES};
// Convert into PPM
enumIGResolutionUnits NewUnits = IG_RESOLUTION_METERS;
IG_util_resolution_units_convert(&Resolution, NewUnits);
```

1.3.1.2.33.12 IG_util_version_get

This utility returns text information about the ImageGear version.

Declaration:

```
VOID ACCUAPI IG_util_version_get(
        [OUT] LPCHAR lpStr
);
```

Arguments:

Name	Type	Description
lpStr	LPCHAR	Pointer to an array of chars where necessary string is returned.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
CHAR szBuf[256];
 ...
IG_util_version_get( szBuf );
 ...
```

1.3.1.2.34 Vector Functions

This section provides information about the Vector group of functions.

- IG_vector_data_get
- IG_vector_data_to_dib
- IG_vector_page_create

1.3.1.2.34.1 IG_vector_data_get

This function returns the vector data interface in the lplpVectorData parameter for the given HIGEAR handle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_vector_data_get(
        HIGEAR hIGear,
        LPVOID* lplpVectorData
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image.
lplpVectorData	LPVOID*	Vector data.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB – 8bpp;
RGB – 24 bpp.

Example:

```
/* C++ interface   */
IIGVectorData*   pVectorData;
AT_ERRCOUNT      nErrCount;
nErrCount = IG_vector_data_get(hIGear, (LPVOID*)& pVectorData);
        if(!nErrCount && pVectorData)
        {
        /* Rotate camera to 90 degrees in XY plane */
        pVectorData-> CameraRotate(3.14159/4, IG_DIR_XY);
        }
/* C interface */
IIGVectorData*    pVectorData;
AT_ERRCOUNT       nErrCount;
nErrCount = IG_vector_data_get(hIGear, (LPVOID*)& pVectorData);
        if(!nErrCount && pVectorData)
        {
        /* Rotate camera to 90 degrees in XY plane */
        pVectorData-> lpVtbl->CameraRotate(pVectorData, 3.14159/4, IG_DIR_XY);
        }
```

Remarks:

The second parameter should be converted to the IIGVectorData pointer to become an accessible for the vector data functionality.

This function can be used only for the PDF and PS vector formats with the ImageGear PDF Component attached only.

1.3.1.2.34.2 IG_vector_data_to_dib

This function flushes the vector data to DIB.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_vector_data_to_dib(
        HIGEAR hIGearSource,
        LPHIGEAR lphIGearDest
);
```

Arguments:

Name	Type	Description
hIGearSource	HIGEAR	Source image handle.
lphIGearDest	LPHIGEAR	New image - non-zero pointer to HIGEAR that takes a raster HIGEAR.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

Indexed RGB – 8bpp;
RGB – 24 bpp.

Vector

Example:

```
HIGEAR rasterhigear;
IG_vector_data_to_dib(hIGear, &rasterhigear);
IG_image_delete(hIGear);
hIGear = rasterhigear;
```

Remarks:

The second parameter must be a non-zero pointer to HIGEAR that takes a raster HIGEAR. The source HIGEAR should be deleted manually.

1.3.1.2.34.3 IG_vector_page_create

This function creates new vector page with empty vector data.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_vector_page_create(
   HIGDIBINFO hDIB,
   HIGEAR* lphIGear
);
```

Arguments:

Name	Type	Description
hDIB	HIGDIBINFO	HIGDIBINFO used to populate the properties of the to-be-created vector page.
lphIGear	HIGEAR*	A reference to the HIGEAR where the vector page will be created.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

To determine the number of errors currently on the error stack use IG_error_check. After fetching all error information you need using IG_error_get, use IG_error_clear to clear the stack.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.2.35 Version Functions

This section provides information about the Version group of functions.

- IG_version_compile_date
- IG_version_numbers

1.3.1.2.35.1 IG_version_compile_date

This function can be called to obtain the date of compilation of the version of ImageGear you are using.

Declaration:

```
LPSTR ACCUAPI IG_version_compile_date (VOID);
```

Return Value:

Returns a FAR pointer to a string containing the compile date in the format given above. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
#include <string.h>
char      MyCompileDateString[12]; /* Will receive "Mmm dd yyyy" */
strcpy ( MyCompileDateString, IG_version_compile_date())
```

Remarks:

The return value is a FAR pointer to a string in the form "Mmm dd yyyy", such as "Jul 04 2010."

1.3.1.2.35.2 IG_version_numbers

This function returns three integers telling you the version of ImageGear that your application is currently using, including the last update installed.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_version_numbers (
        LPINT lpVerMajor,
        LPINT lpVerMinor,
        LPINT lpVerUpdate
);
```

Arguments:

Name	Type	Description
lpVerMajor	LPINT	Far pointer to an INT variable in which will be stored the Major version number of the version of ImageGear that you are using.
lpVerMinor	LPINT	Far pointer to an INT variable in which will be stored the Minor version number of the version of ImageGear that you are using.
lpVerUpdate	LPINT	Far pointer to an INT variable in which will be stored the Update (bug fix) number, reflecting any updates you have received and installed in this version of ImageGear.

Return Value:

Returns the number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
INT         nVerMaj, nVerMin, nVerUpdate;
IG_version_numbers ( &nVerMaj, &nVerMin, &VerUpdate );
```

Remarks:

The Major and Minor version numbers are, for example, the "10" and "0" respectively, for the first release of ImageGear 10.0 (ImageGear2000).

The above numbers appear in accucnst.h. Your application can compare the accucnst.h constants with the numbers returned here to verify that the proper version of the ImageGear DLL is being loaded and used.

1.3.1.2.36 Warning Functions

This section provides information about the Warning group of functions.

- IG_warning_check
- IG_warning_clear
- IG_warning_get
- IG_warning_set

1.3.1.2.36.1 IG_warning_check

This function returns the number of warnings currently on the ImageGear error stack.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_warning_check();
```

Arguments:

None

Return Value:

Returns the number of warnings on the error stack. If errors occur during this function call, the function returns (AT_ERRCOUNT)-1, but these errors are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

A call to this function has the same effect as a call to IG_err_error_check with nLevel equal to 1.

1.3.1.2.36.2 IG_warning_clear

This function clears all warnings from the error stack.

Declaration:

```
VOID ACCUAPI IG_warning_clear();
```

Arguments:

None

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
IG_warning_clear();
```

Remarks:

After calling this function, IG_warning_check will return zero.

1.3.1.2.36.3 IG_warning_get

This function retrieves an ImageGear warning Code and associated information from the error stack.

Declaration:

```
VOID ACCUAPI IG_warning_get(
   INT iErrorIndex,
   LPSTR szFileName,
   INT cbFileNameSize,
   LPINT lpiLineNumber,
   LPAT_ERRCODE lpiCode,
   LPAT_INT lplValue1,
   LPAT_INT lplValue2,
   LPSTR pszWarning,
   INT nWarningSize
);
```

Arguments:

Name	Type	Description
iErrorIndex	INT	Tells which warning to fetch from stack. A value of 0 means fetch the first warning placed on the stack.
szFileName	LPSTR	Pointer indicating where to return the module name in which this error occurred. If this pointer is NULL, the module name is not returned.
cbFileNameSize	INT	Number of bytes available in byte array pointed to by szFileName.
lpiLineNumber	LPINT	Pointer indicating where to return the line number at which the warning occurred. If NULL, the line number is not returned.
lpiCode	LPAT_ERRCODE	Pointer indicating where to return the Warning Code. If NULL, the Warning Code is not returned.
lplValue1	LPAT_INT	Pointer indicating where to return a value stored as lValue1 when the warning occurred. If NULL, this value is not returned. See Remarks below for explanation of lValue1 and lValue2.
lplValue2	LPAT_INT	Pointer indicating where to return a value stored as lValue2 when the warning occurred. If NULL, this value is not returned. See Remarks below for explanation of lValue1 and lValue2.
pszWarning	LPSTR	Pointer indicating where to return additional text description.
nWarningSize	INT	Size of the memory buffer pszWarning.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
INT i;                          // Will hold Loop Index and Error Index
INT iLineNumber;              // Will hold returned Line Number
BYTE szFileName[30];          // Will hold ret'd module name, up to 29 chars
INT cbFileNameSize;             // Will hold size of szFileName array
AT_INT lValue1, lValue2;      // Will hold returned lValue1, lValue2
AT_ERRCODE iWarnCode;          // Will hold returned warning code
CHAR szWaringMessage[256];      // Will hold returned warning message
AT_ERRCOUNT nWarnCount;          // Will hold count of errors on error stack
```

```
TCHAR szBuf[60];            // Will hold zero-terminated string returned by wsprintf()
below
cbFileNameSize = 30;        // Size of module-name array
nWarnCount = IG_warning_check(); // Get number of errors on stack
for ( i = 0;  i < nWarnCount;  i++ )
{
    // Get Module Name, Line Number, Error Code, and lValue1, lValue2:
    IG_warning_get ( i, (LPSTR) &szFileName,
            cbFileNameSize, &iLineNumber, (LPAT_ERRCODE)&iWarnCode,
            (LPAT_INT) &lValue1, (LPAT_INT) &lValue2,
            szWaringMessage, sizeof(szWaringMessage));
    // Format warning message in szBuf:
    wsprintf ( szBuf, _T("Earning %d in Module %s at Line %d"), iWarnCode, szFileName,
iLineNumber );
    // Display warning message in a Message Box, with heading "Warning" :
    MessageBox ( NULL, szBuf, _T("Warning"), MB_OK );
}
IG_warning_clear(); // Done getting errors, clear the error stack
```

Remarks:

Set iErrorIndex to indicate which warning to get. iErrorIndex = 0 means the warning added to the stack first. The other arguments (except cbFileNameSize) are pointers telling this function where to return the retrieved information to you. This information consists of the Warning Code, the module name and line number at which the error occurred, and two additional values (lValue1 and lValue2) that may provide additional information about the warning and a buffer for the additional text information. A size of the buffer is passed in the last parameter. See the Appendix for a list of all ImageGear Error Codes and the significance of lValue1, lValue2 where applicable.

To determine the number of warnings currently on the error stack use IG_warning_check. After fetching all error information you need using IG_warning_get, use IG_warning_clear to clear the stack.

A call to this function has the same effect as a call to IG_err_error_get with nLevel equal to 1. If the user has defined his own warning levels (greater than 1), he should use IG_err_error_get function instead.

1.3.1.2.36.4 IG_warning_set

This function places an ImageGear warning onto the error stack.

Declaration:

```
AT_ERRCODE ACCUAPI IG_warning_set(
   const LPSTR szFileName,
   INT iLineNumber,
   AT_ERRCODE iCode,
   AT_INT lValue1,
   AT_INT lValue2,
   const LPSTR szWarning
);
```

Arguments:

Name	Type	Description
szFileName	const LPSTR	Pointer to a string that supplies the name of the module from which the warning was generated. It is recommended that you use the _FILE_ constant in this field.
iLineNumber	INT	An integer telling ImageGear from which line the warning was set. It is recommended that you use the _LINE_ constant in this field.
iCode	AT_ERRCODE	An integer value of type AT_ERRCODE. Set this to the code number of the warning that you wish to place on the error stack.
lValue1	AT_INT	The first argument that supplies any supporting information about the warning. Your application might use this value to decide what to do after setting a particular kind of warning.
lValue2	AT_INT	The second argument that supplies any supporting information about the warning. Your application might use this value to decide what to do after setting a particular kind of warning.
szWarning	const LPSTR	Additional text description of the warning. It can be NULL if it is not available.

Return Value:

Returns the code of the ImageGear error that occurred during this function call. A value of zero means no errors have occurred. Errors that occurred during this function call are not appended onto the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
static const AT_ERRCODE MYWARNING = (IGE_LAST_ERROR_NUMBER - 2);
AT_ERRCOUNT nErrcount = IG_warning_set(__FILE__, __LINE__, MYWARNING, 0, 0, "Warning
message");
```

Remarks:

If you are setting a warning code that you have defined yourself, you must make sure that it has a value less than ImageGear's IGE_LAST_ERROR_NUMBER. As the defined value of IGE_LAST_ERROR_NUMBER may change in the future, you should define your warning codes relatively to IGE_LAST_ERROR_NUMBER, as demonstrated in the example, rather than use literal values.

1.3.1.3 Core Component Callback Functions Reference

This section provides information about the ImageGear Core Component callback functions, which are organized alphabetically.

- LPAFT_IG_ICC_GET_CB
- LPAFT_IG_METAD_ITEM_ADD_CB
- LPAFT_IG_METAD_ITEM_GET_CB
- LPAFT_IG_METAD_ITEM_SET_CB
- LPFNIG_BATCH_BEFORE_OPEN
- LPFNIG_BATCH_BEFORE_SAVE
- LPFNIG_DIB_CREATE
- LPFNIG_DIB_CREATE_EX
- LPFNIG_DIB_GET
- LPFNIG_DIB_GET_EX
- LPFNIG_DIRECT_RASTER_GET
- LPFNIG_ERRMNGR_ADD
- LPFNIG_ERRMNGR_CLEAR
- LPFNIG_ERRSTACK_ADD
- LPFNIG_ERRSTACK_CLEAR
- LPFNIG_IMAGESPOOLED
- LPFNIG_LOAD_DISP
- LPFNIG_MEM_ALLOC
- LPFNIG_MEM_FREE
- LPFNIG_MEM_REALLOC
- LPFNIG_MPCB_UPDATE
- LPFNIG_RASTER_PLANE_SET
- LPFNIG_RASTER_GET
- LPFNIG_RASTER_SET
- LPFNIG_READ
- LPFNIG_SEEK
- LPFNIG_SIZE_CHANGE
- LPFNIG_STATUS_BAR
- LPFNIG_TAG_GET
- LPFNIG_TAG_SET
- LPFNIG_TAG_USER_GET
- LPFNIG_WRITE

1.3.1.3.1 LPAFT_IG_ICC_GET_CB

This callback function is called every time the format filter encounters an ICC profile in the loaded image.

Declaration:

```
typedef VOID (LPACCUAPI  LPAFT_IG_ICC_GET_CB)(
        AT_VOID* lpPrivate,
        AT_BYTE* lpICCData,
        AT_INT DataLength
);
```

Arguments:

Name	Type	Description
lpPrivate	AT_VOID*	Private callback data.
lpICCData	AT_BYTE*	ICC profile data, allocated by ImageGear.
DataLength	AT_INT	Length of ICC profile data, in bytes.

Return Value:

N/A

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The ICC profile is provided in the standard ICC format as a byte array. ImageGear allocates the buffer for ICC profile data so that the application does not delete it. If the application needs to use the ICC profile after the callback call, it copies it to its own buffer.

Use IG_fltr_ICC_callback_set() to register this callback function.

Use IG_fltr_ICC_callback_get() function to retrieve your ICC callback settings.

1.3.1.3.2 LPAFT_IG_METAD_ITEM_ADD_CB

This callback function is used to insert new items during a WRITE filter operation.

Declaration:

```
typedef AT_BOOL (LPACCUAPI LPAFT_IG_METAD_ITEM_ADD_CB)(
        LPVOID lpPrivate,
        AT_MODE FormatID,
        LPCHAR ItemName,
        DWORD ItemID,
        AT_MODE ItemType,
        LPVOID ItemValue,
        AT_MODE ValueType,
        DWORD ValueLength,
        AT_BOOL ReadOnlyValue
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Private callback data.
FormatID	AT_MODE	The ID of the format filter that will send or get the item (IG_FORMAT_... constant).
ItemName	LPCHAR	Provides the name of the new item.
ItemID	DWORD	Numerical ID of the item.
ItemType	AT_MODE	Specifies the type of the item and reflects the status of the given record. Possible values are: • IG_METAD_VALUE_ITEM - this value specifies that the current item is a value of the simplest type, and the field Value contains the actual value of the item, and ValueType contains the identifier of the type of this item. ReadOnly can be either TRUE (readonly) or FALSE (read/ write). The Name and/or Id contains textual and numerical identification of the item. • IG_METAD_LEVEL_START - this value specifies that the current item opens a sublevel of items, and all next items up to the corresponding item with a LEVEL_END value belong to this sublevel. • IG_METAD_LEVEL_END - this value closes the current sublevel and indicates that the next item belongs to a higher level.
ItemValue	LPVOID	Value of the new item. If ItemType = IG_METAD_VALUE_ITEM then ItemValue contains the actual value of the item of type specified by the ValueType parameter. The ItemValue is stored as an array of elements where each element contains values of type ValueType. Length of array is provided in parameter ValueLength.
ValueType	AT_MODE	If ItemType = IG_METAD_VALUE_ITEM then contains actual type of value stored in ItemValue pointer. See the Non-Image Data Format section for exact list of possible types.
ValueLength	DWORD	Length of array in ItemValue.
ReadOnlyValue	AT_BOOL	This parameter is not used.

Return Value:

Returns TRUE if a new value is added to item data; FALSE if it is not. A return of TRUE will cause the call of the Add callback function one more time.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

See example for IG_fltr_metad_callback_set() function.

Remarks:

If the return value is TRUE then a new value is added; and if FALSE a new value is not added. ImageGear will be calling Add callback function while it returns TRUE, so the callback function should return TRUE if the provision of additional items is it not finished yet, and FALSE if it is finished. All parameters except of lpPrivate are used to get information about the new value.

ImageGear assumes that all tags passed via LPAFT_IG_METAD_ITEM_ADD_CB callback are writable, except for the tags that were sent by LPAFT_IG_METAD_ITEM_SET_CB, marked as read-only.

See also IG_fltr_metad_callback_get(), IG_fltr_metad_callback_set(), LPAFT_IG_METAD_ITEM_GET_CB, LPAFT_IG_METAD_ITEM_SET_CB functions and the section Using Filter Callback Functions to Process Non-Image Data.

1.3.1.3.3 LPAFT_IG_METAD_ITEM_GET_CB

This callback function is used to get information about metadata items received during a READ filter operation.

Declaration:

```
typedef VOID (LPACCUAPI LPAFT_IG_METAD_ITEM_GET_CB)(
        LPVOID lpPrivate,
        AT_MODE FormatID,
        LPCHAR ItemName,
        DWORD ItemID,
        AT_MODE ItemType,
        LPVOID ItemValue,
        AT_MODE ValueType,
        DWORD ValueLength,
        AT_BOOL ReadOnlyValue
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Private callback data.
FormatID	AT_MODE	The ID of the format filter that will send or get the item (IG_FORMAT_... constant).
ItemName	LPCHAR	Name of the item.
ItemID	DWORD	Numerical ID of the item.
ItemType	AT_MODE	Specifies the type of the item and reflects the status of the given record. Possible values are: • IG_METAD_VALUE_ITEM - this value specifies that the current item is a value of the simplest type, and the field Value contains the actual value of the item, and ValueType contains the identifier of the type of this item. ReadOnly can be either TRUE (read-only) or FALSE (read/write). Name and/or Id contains textual and numerical identification of item. • IG_METAD_LEVEL_START - this value specifies that the current item opens a sublevel of items, and all next items up to corresponding item with a IG_METAD_LEVEL_END value belong to this sublevel. • IG_METAD_LEVEL_END - this value closes the current sublevel and indicates that the next item belongs to a higher level.
ItemValue	LPVOID	If ItemType = IG_METAD_VALUE_ITEM then ItemValue contains the actual value of the item of the type specified by the ValueType parameter. Value is stored as array of elements where each element contains values of type ValueType. Length of array is provided in parameter ValueLength.
ValueType	AT_MODE	Type of element stored in array ItemValue.
ValueLength	DWORD	Length of array of elements stored in ItemValue.
ReadOnlyValue	AT_BOOL	If this argument is TRUE, then the actual value of the item cannot be changed by the callback function, and the value is passed for informational purposes only. If FALSE then the value of the item can be changed, and the application can provide a new value through the next three parameters.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

See example for IG_fltr_metad_callback_set() function.

Remarks:

The ReadOnlyValue parameter is used to inform the application that the value of a given item is for information only and cannot be changed during future operations.

See also IG_fltr_metad_callback_get(), IG_fltr_metad_callback_set(), LPAFT_IG_METAD_ITEM_ADD_CB, LPAFT_IG_METAD_ITEM_SET_CB functions and the section Using Filter Callback Functions to Process Non-Image Data.

1.3.1.3.4 LPAFT_IG_METAD_ITEM_SET_CB

If this callback function is defined in the format filter, then during a filter WRITE operation it is called every time some data is ready to be written.

Declaration:

```
typedef AT_BOOL (LPACCUAPI LPAFT_IG_METAD_ITEM_SET_CB)(
        LPVOID lpPrivate,
        AT_MODE FormatID,
        LPCHAR ItemName,
        DWORD ItemID,
        AT_MODE ItemType,
        LPVOID ItemValue,
        AT_MODE ValueType,
        DWORD ValueLength,
        AT_BOOL ReadOnlyValue,
        LPVOID* NewItemValue,
        LPAT_MODE* NewValueType,
        LPDWORD* NewValueLength
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Private callback data associated with the metadata callback function.
FormatID	AT_MODE	The ID of the format filter that will send or get the item (IG_FORMAT_... constant).
ItemName	LPCHAR	Text name of the item.
ItemID	DWORD	Numerical ID of the item.
ItemType	AT_MODE	Specifies the type of the item and reflects the status of a given record. Possible values are: • IG_METAD_VALUE_ITEM - this value specifies that the current item is a value of the simplest type, and the field Value contains the actual value of the item, and the ValueType contains the identifier of the type of this item. ReadOnly can be either TRUE (read-only) or FALSE (read/write). The Name and/or Id contains the textual and numerical identification of item. • IG_METAD_LEVEL_START - this value specifies that the current item opens a sublevel of items, and all the next items up to the corresponding item with a LEVEL_END value belong to this sublevel. • IG_METAD_LEVEL_END - this value closes the current sublevel and indicates that the next item belongs to a higher level.
ItemValue	LPVOID	If ItemType = IG_METAD_VALUE_ITEM then this argument contains the actual value of the item of the type specified by the ValueType parameter. Value is stored as an array of elements where each element contains values of type ValueType. The Length of array is provided in the parameter ValueLength.
ValueType	AT_MODE	If ItemType = IG_METAD_VALUE_ITEM then ValueType contains the actual type of value stored in the ItemValue pointer. See Non-Image Data Format for the exact list of possible types.
ValueLength	DWORD	If ItemType = IG_METAD_VALUE_ITEM then ValueLength contains the number of elements in array ValueItem of the type specified by ValueType.
ReadOnlyValue	AT_BOOL	If this argument is TRUE, then the actual value of the item cannot be changed by the callback function, and the value is passed for informational purposes only. If FALSE, then the value of item can be changed, and the application can provide the new value through the next three parameters.
NewItemValue	LPVOID*	Pointer to the new item value that the application may request to set as a replacement of the data passed in the ItemValue parameter. If it is not NULL,

		then the next two parameters contain the type of the value and the length of the array of elements.
NewValueType	LPAT_MODE*	Specifies the type of the new item value passed through the NewItemValue parameter.
NewValueLength	LPDWORD*	Specifies the size of the array passed through the NewItemValue pointer.

Return Value:

Returns TRUE if ImageGear should overwrite the default item data with your data, or FALSE if ImageGear should ignore your data.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

See example for IG_fltr_metad_callback_set() function.

Remarks:

The application can change/provide a new value for the given item using NewItemValue, NewValueType, and NewValueLength parameters. This new value is to be used as a replacement for the default data passed through the ItemValue parameter if ReadOnlyValue is FALSE. The callback function should return TRUE if the callback function has changed value, and FALSE if it has not changed.

See also IG_fltr_metad_callback_get(), IG_fltr_metad_callback_set(), LPAFT_IG_METAD_ITEM_ADD_CB, LPAFT_IG_METAD_ITEM_GET_CB functions and the section Using Filter Callback Functions to Process Non-Image Data.

1.3.1.3.5 LPFNIG_BATCH_BEFORE_OPEN

LPFNIG_BATCH_BEFORE_OPEN is called before a file is opened, allowing you to get the file name and correct some settings.

Declaration:

```
typedef AT_BOOL (LPACCUAPI LPFNIG_BATCH_BEFORE_OPEN)(
        LPVOID lpPrivate,
        const LPSTR lpszFileName
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	A far pointer to private data area.
lpszFileName	const LPSTR	The name of file to be opened.

Return Value:

Reserved (must always be TRUE).

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
//Dll
//User's BatchBeforeOpen CB
AT_BOOL MyBatchBeforeOpen(
   LPVOID         lpPrivate,     /* Private data passed in*/
                const LPSTR    lpszFileName   /* File name to be open*/
   )
{
// Set PDF control name if the file is PDF
                AT_MODE nFileType;
                IG_info_get(lpszFileName, 1, NULL, nFileType, NULL, NULL);
                if(nFileType == IG_FORMAT_PDF)
                {
                        IG_fltr_ctrl_set(IG_FORMAT_PDF, "FILENAME", (LPVOID) lpszFileName,
sizeof(lpszFileName));
                }
                return TRUE;
}
// Register BatchBeforeOpen CB
IG_batch_CB_register(MyBatchBeforeOpen, IG_BATCHCB_BEFORE_OPEN, NULL);
```

Remarks:

For example, some multimedia formats and PDF file names require you to get the file name before page conversion.

1.3.1.3.6 LPFNIG_BATCH_BEFORE_SAVE

LPFNIG_BATCH_BEFORE_SAVE is called before an image file is saved, allowing you to correct and image before saving.

Declaration:

```
typedef AT_BOOL (LPACCUAPI LPFNIG_BATCH_BEFORE_SAVE)(
        LPVOID lpPrivate,
        HIGEAR hIGear,
        UINT nPageNumber
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area.
hIGear	HIGEAR	HIGEAR handle to the image.
nPageNumber	UINT	This variable is set to the number of pages to be saved.

Return Value:

Reserved (must always be TRUE).

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
// DLL
// User's BatchBeforeSave CB
AT_BOOL MyBatchBeforeSave(
        LPVOID         lpPrivate,    /* Private data passed in     */
        HIGEAR         hIGear,                /* ImageGear image handle     */
        UINT           nPageNumber            /* Number of page to be saved */
   )
{
        // Convert bpp to 1 before saving
        UINT bits_per_pixel;
        IG_image_dimensions_get(hIGear, NULL, NULL, & bits_per_pixel);
        If(bits_per_pixel != 1)
        {
                IG_IP_color_reduce_bayer(hIGear, 1, NULL);
}
        return TRUE;
}
// Register BatchBeforeSave CB
IG_batch_CB_register(MyBatchBeforeSave, IG_BATCHCB_BEFORE_SAVE, this);
```

Remarks:

For example, you might want to rotate an image before saving.

Multipage documents can be saved as a set of different files if the flag IG_BATCH_MP_TO_MP is not specified.

1.3.1.3.7 LPFNIG_DIB_CREATE

This is one of the two types of callback functions supplied in calls to IG_load_FD_CB() and IG_load_mem_CB().

Declaration:

```
typedef AT_ERRCOUNT (ACCUAPI LPFNIG_DIB_CREATE) (
        LPVOID lpPrivate,
        const LPAT_DIB lpDIB,
        const LPAT_RGBQUAD lpRGB
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area.
lpDIB	const LPAT_DIB	Far pointer to start of an AT_DIB DIB header (that is, a BITMAPINFOHEADER) struct created by ImageGear.
lpRGB	const LPAT_RGBQUAD	Far pointer to the first of the AT_RGBQUAD structs constituting the palette in the DIB. Will be NULL if the image is 24-bit.

Return Value:

Returns an error count.

Supported Raster Image Formats:

Indexed RGB – 1…8 bpp;
Grayscale – 9…16 bpp;
RGB – 24 bpp;
CMYK – 32 bpp.

This callback function is only kept for backward compatibility reasons. Please use IG_load_FD_CB_ex / IG_load_mem_CB_ex and LPFNIG_DIB_CREATE_EX instead.

Example:

```
BOOL  ACCUAPI MyDIB_Create (LPVOID lpPrivate, LPAT_DIB lpDIB, LPAT_RGBQUAD lpRGB )
{
/* Can allocate memory, create a DIB header (AT_DIB) and palette. Later, an
LPFNIG_RASTER_SET function can create the image bitmap. */
 ...
return IG_error_check();
}
```

Remarks:

This callback function is called by ImageGear to provide your application the information it needs to create its own DIB header and palette.

On entry to this function lpDIB points to an AT_DIB struct which ImageGear has created upon reading the file's header. You can use information from this AT_DIB struct to create your own DIB header, but you should not alter the information at lpDIB.

Similarly, lpRGB points to the palette as obtained from the file. (lpRGB = NULL if no palette.) You can copy the palette, or create your own for the DIB you are creating.

If you need to terminate the load, you can place an error on the stack yourself, using IG_error_set(). See the description for that function.

1.3.1.3.8 LPFNIG_DIB_CREATE_EX

This callback function is called after the image header has been read.

Declaration:

```
typedef AT_ERRCOUNT (LPACCUAPI LPFNIG_DIB_CREATE_EX)(
        LPVOID lpPrivate,
        HIGDIBINFO hDIB
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area.
hDIB	HIGDIBINFO	Extended DIB header.

Return Value:

Returns an error count.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

It passes image information, including palette, through HIGDIBINFO structure.

hDIB object is owned by ImageGear. The application shall not use it after exiting from the callback function, and shall not delete it.

1.3.1.3.9 LPFNIG_DIB_GET

This is one of the two types of callback functions supplied in calls to IG_save_FD_CB_ex()and IG_save_mem_CB_ex().

Declaration:

```
typedef AT_ERRCOUNT (ACCUAPI LPFNIG_DIB_GET) (
        LPVOID lpPrivate,
        LPAT_DIB lpDIB,
        LPAT_RGBQUAD lpRGB
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area.
lpDIB	LPAT_DIB	Far pointer to start of the DIB header, that is the AT_DIB (BITMAPINFOHEADER) struct that begins the DIB.
lpRGB	LPAT_RGBQUAD	Far pointer to first of the AT_RGBQUAD structs constituting the palette in the DIB. Will be NULL if it is a 24-bit image.

Return Value:

Returns an error count.

Supported Raster Image Formats:

Indexed RGB – 1…8 bpp;
Grayscale – 9…16 bpp;
RGB – 24 bpp;
CMYK – 32 bpp.

This callback function is only kept for backward compatibility reasons. Please use IG_save_FD_CB_ex / IG_save_mem_CB_ex and LPFNIG_DIB_GET_EX instead.

Example:

```
BOOL  ACCUAPI MyDIBGet (LPVOID lpPrivate, LPAT_DIB lpDIB, LPAT_RGBQUAD lpPalette )
{
/* Modify the DIB header fields at *lpDIB as desired, and store a
   palette at *lpPalette */
 ...
return IG_error_check();
}
```

Remarks:

This callback function is called by ImageGear prior to saving the DIB header and palette.

On entry to this function, lpDIB points to the image's DIB header (AT_DIB or BITMAPINFOHEADER struct), and lpRGB points to its DIB palette. This function is responsible for setting the DIB header fields (width, height, bits per pixel, compression, etc.) and for assuring the palette desired if it is not a 24-bit image.

If you need to terminate the load, you can place an error on the stack yourself, using IG_error_set(). See the description for that function.

1.3.1.3.10 LPFNIG_DIB_GET_EX

This callback function is called before writing the image header.

Declaration:

```
typedef AT_ERRCOUNT (LPACCUAPI LPFNIG_DIB_GET_EX)(
        LPVOID lpPrivate,
        HIGDIBINFO* lphDIB
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area.
lphDIB	HIGDIBINFO*	Extended DIB header.

Return Value:

Returns an error count.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

The application shall create a new HIGDIBINFO object containing information about the image, and pass through the lphDIB parameter. ImageGear owns the HIGDIBINFO object after exiting from the callback, and eventually deletes it. If the image being saved has a palette, it should be passed with HIGDIBINFO as well.

1.3.1.3.11 LPFNIG_DIRECT_RASTER_GET

This function has been deprecated and will be removed from the public API in a future release.

Declaration:

```
typedef LPAT_PIXEL (LPACCUAPI LPFNIG_DIRECT_RASTER_GET)(
        LPVOID lpPrivate,
        AT_PIXPOS cyPos,
        DWORD cRasterSize,
        LPAT_ERRCODE lpnErrCode
);
```

Arguments:

Name	**Type**	**Description**
lpPrivate	LPVOID	Not used.
cyPos	AT_PIXPOS	Not used.
cRasterSize	DWORD	Not used.
lpnErrCode	LPAT_ERRCODE	Not used.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.1.3.12 LPFNIG_ERRMNGR_ADD

This callback is called each time a new error record is added by any thread.

Declaration:

```
typedef VOID (LPACCUAPI LPFNIG_ERRMNGR_ADD)(
        LPVOID lpPrivate,
        DWORD dwThreadID,
        UINT nRecord,
        INT iLineNumber,
        AT_ERRCODE  iCode,
        UINT nLevel,
        AT_INT lValue1,
        AT_INT lValue2,
        LPCHAR lpFileName,
        LPCHAR lpExtratext
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Private data passed.
dwThreadID	DWORD	Thread identifier where error happened.
nRecord	UINT	Index of this record in the stack.
iLineNumber	INT	Line number where a problem occurred.
iCode	AT_ERRCODE	Error code.
nLevel	UINT	Level of the error.
lValue1	AT_INT	Specific value identifying the reason for an error.
lValue2	AT_INT	Specific value identifying the reason for an error.
lpFileName	LPCHAR	Pointer to a string holding a filename or NULL if not available.
lpExtratext	LPCHAR	Pointer to a string holding extra info or NULL if not available.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
VOID ACCUAPI ErrGlAddCB(
        LPVOID   lpPrivate,      /* Private data passed in.  */
        DWORD    dwThreadID,    /* Thread identifier where record added.     */
        UINT     nRecord,      /* index of this record in the stack.        */
        INT      iLineNumber,   /* line number where problen occurred.       */
        AT_ERRCODE       iCode,      /* error code.        */
        UINT      nLevel,     /* level of the error. */
        LONG    lValue1,
        LONG     lValue2,
        LPCHAR   lpFileName,    /* filename str ofr NULL if not present. */
        LPCHAR   lpExtratext    /* extra text info about error. */
);
```

```
{
        char     szOutput[1024];
        sprintf( szOutput, "Global CallBack - new error record
added:\nThread=%i\nRecord=%i\nLine=%i\nCode=%i\nLevel=%i\nValue1=%i; Value2=%i,\nFile
Name: %s\nExtra Text: %s",
                dwThreadID, nRecord, iLineNumber, iCode, nLevel, lValue1, lValue2,
lpFileName,
lpExtratext );
        //AfxMessageBox( szOutput );
        ::MessageBox(NULL, szOutput, "THREADS", MB_OK);
}
```

1.3.1.3.13 LPFNIG_ERRMNGR_CLEAR

This callback is called each time an error stack is cleared by any thread.

Declaration:

```
typedef VOID (LPACCUAPI LPFNIG_ERRMNGR_CLEAR)(
        LPVOID lpPrivate,
        DWORD dwThreadID,
        UINT nRecords
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Private data passed.
dwThreadID	DWORD	Thread identifier where the stack cleared.
nRecords	UINT	Number of records cleared from the stack starting from index 0.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
VOID ACCUAPI ErrGlClearCB(
LPVOID    lpPrivate,    /* Private data passed in. */
DWORD     dwThreadID,   /* Thread identifier where stack cleared.  */
UINT      nRecords    /* Number of records cleared from the stack starting from 0 index. */
)
{
char      szOutput[1024];
sprintf( szOutput, "Global CallBack - error stack cleared\nThread:%i, Records cleared:%i",
dwThreadID, nRecords );
        //AfxMessageBox( szOutput );
        ::MessageBox(NULL, szOutput, "THREADS", MB_OK);
}
```

Remarks:

Since each thread has its own independent error stack clearing, a stack by one thread does not cause other stacks to clear.

1.3.1.3.14 LPFNIG_ERRSTACK_ADD

This callback function is called each time a thread that registered this callback (using IG_err_callback_set() function) adds a new record to the error stack.

Declaration:

```
typedef VOID (LPACCUAPI LPFNIG_ERRSTACK_ADD)(
        LPVOID lpPrivate,
        UINT nRecord,
        INT iLineNumber,
        AT_ERRCODE iCode,
        UINT nLevel,
        AT_INT lValue1,
        AT_INT lValue2,
        LPCHAR lpFileName,
        LPCHAR lpExtratext
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Private data passed in when you register a callback.
nRecord	UINT	Index of this record in the stack.
iLineNumber	INT	Line number where the problem has occurred.
iCode	AT_ERRCODE	Error code.
nLevel	UINT	Level of the error.
lValue1	AT_INT	Specific value identifying the reason for the error.
lValue2	AT_INT	Specific value identifying the reason for the error.
lpFileName	LPCHAR	Pointer to a string holding a filename or NULL if not available.
lpExtratext	LPCHAR	Pointer to a string holding additional information, or NULL if not available.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
VOID   ACCUAPI  LocThErrAddCB(
LPVOID   lpPrivate,    /* Private data passed in.  */
UINT    nRecord,     /* index of this record in the stack    */
INT      iLineNumber, /* line number where problen occurred   */
AT_ERRCODE   iCode,      /* error code. */
UINT      nLevel,        /* level of the error.     */
LONG      lValue1,
LONG      lValue2,
LPCHAR    lpFileName,  /* filename str ofr NULL if not present. */
LPCHAR    lpExtratext  /* extra text info about error.  */
)
{
        char     szOutput[1024];
        sprintf( szOutput, "Local CallBack - new error record added:\nThread Id:%u\nThread
Number=%i\nRecord=%i\nLine=%i\nCode=%i\nLevel=%i\nValue1=%i; Value2=%i,\nFile Name:
%s\nExtra Text: %s",
                GetCurrentThreadId(), (int)lpPrivate, nRecord, iLineNumber, iCode, nLevel,
lValue1,
lValue2, lpFileName, lpExtratext );
        //AfxMessageBox( szOutput );
        ::MessageBox(NULL, szOutput, "THREADS", MB_OK);
}
```

1.3.1.3.15 LPFNIG_ERRSTACK_CLEAR

This callback function is called each time a thread that registered this callback (using IG_err_callback_set() function) clears the stack.

Declaration:

```
typedef VOID (LPACCUAPI LPFNIG_ERRSTACK_CLEAR)(
        LPVOID lpPrivate,
        UINT nRecords
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Private data passed.
nRecords	UINT	Number of records cleared from the stack starting with index 0.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
VOID  ACCUAPI  LocThErrClearCB(
LPVOID   lpPrivate,     /* Private data passed in.                   */
UINT     nRecords     /* Number of records cleared from the stack starting from 0 index. */
)
{
        char     szOutput[1024];
        sprintf( szOutput, "Local CallBack - error stack cleared\nThread Id:%u\nThread
number:%i, Records cleared:%i",
                GetCurrentThreadId(), (int)lpPrivate, nRecords );
        // AfxMessageBox( szOutput );
        ::MessageBox(NULL, szOutput, "THREADS", MB_OK);
}
```

1.3.1.3.16 LPFNIG_IMAGESPOOLED

This function is called by IG_dspl_document_print() to determine which image in the array of images has just been spooled to the printer.

Declaration:

```
typedef BOOL (ACCUAPI LPFNIG_IMAGESPOOLED) (
        LPVOID lpPrivate,
        UINT nImageNumber,
        UINT nPageNumber
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area you provided in your call to IG_dspl_document_print() and IG_dspl_document_print_custom() functions.
nImageNumber	UINT	A variable of type UINT in which you can keep track of the number of hIGear images just processed in the array of images.
nPageNumber	UINT	A variable of type UINT in which you can keep track of the number of the page (of paper) being printed.

Return Value:

This callback function returns TRUE if the image is successfully processed, and FALSE if it is not. As soon as the callback function returns FALSE, ImageGear stops processing the images.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The nPageNumber argument is the current page number being printed. This callback does not need to be "registered" by a call to an IG_...CB_register() function.

1.3.1.3.17 LPFNIG_LOAD_DISP

This function is called by IG_load_file_display() after it has loaded the image and assigned to it a HIGEAR handle, but before it has displayed the image.

Declaration:

```
typedef VOID (ACCUAPI LPFNIG_LOAD_DISP) (
        LPVOID lpPrivate,
        HIGEAR hIGear
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area you provided in your call to IG_load_file_display() function.
hIGear	HIGEAR	HIGEAR handle assigned to the image just loaded.

Return Value:

None

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

See the Example code in the section Working with ImageGear Callback Functions.

Remarks:

Using the HIGEAR handle supplied to you in this call, you can set the image and device rectangles, set display attributes, and perform other operations you choose, prior to returning. Upon your return, IG_load_file_display() will continue and will display your image, line by line, using the settings you have made.

1.3.1.3.18 LPFNIG_MEM_ALLOC

Create a function of this type to give your application the flexibility of replacing ImageGear's memory allocation routine with your own.

Declaration:

```
typedef LPBYTE (ACCUAPI LPFNIG_MEM_ALLOC)(AT_UINT nSize);
```

Arguments:

Name	Type	Description
nSize	AT_UINT	Number of bytes to allocate.

Return Value:

The user supplied callback function should return a pointer to the allocated block of memory.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/***************************************************************/
/* Memory Alloc callback function definition */
/**************************************************************/
LPBYTE  ACCUAPI MyMemAlloc(AT_UINT nSize) /* number of bytes to alloc*/
{
        /* Put your own memory allocation code here */
        return( buffer);
};
/*See also example for IG_mem_CB_register() */
```

Remarks:

This callback function is registered by calling IG_mem_CB_register(). The register function must be called prior to any user-defined callback functions being used by the ImageGear library.

Your memory allocation function will only be used when large allocations (allocations greater than 1024) are performed.

1.3.1.3.19 LPFNIG_MEM_FREE

Create a function of this type to give your application the flexibility of replacing ImageGear's memory free routine with your own.

Declaration:

```
typedef LPBYTE (ACCUAPI LPFNIG_MEM_FREE) ( LPBYTE lpBuffer);
```

Arguments:

Name	Type	Description
lpBuffer	LPBYTE	Far pointer to the buffer to be freed.

Return Value:

Usually NULL.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
/***************************************************************/
/* Memory Free callback function definition       */
**************************************************************/
LPBYTE  ACCUAPI MyMemFree( LPBYTE lpBuffer)
{
        /*MEMORY FREE CODE*/
        return NULL;
};
/*See also example for IG_mem_CB_register() */
```

Remarks:

This callback function is registered by calling IG_mem_CB_register(). The register function must be called prior to any user-defined callback functions being used by the ImageGear library.

1.3.1.3.20 LPFNIG_MEM_REALLOC

Create a function of this type to give your application the flexibility of replacing ImageGear's memory reallocation routine with your own.

Declaration:

```
typedef LPBYTE (ACCUAPI LPFNIG_MEM_REALLOC) (
        LPBYTE lpBuffer,
        AT_UINT nSize
);
```

Arguments:

Name	Type	Description
lpBuffer	LPBYTE	Far pointer to the buffer to be reallocated.
nSize	AT_UINT	New byte count for realloc buffer.

Return Value:

The user supplied callback function should return a pointer to the allocated block of memory.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
***********************************************************/
/* Memory ReAlloc callback function definition        */
***********************************************************/
LPBYTE ACCUAPI MyMemReAlloc( LPBYTE lpbuffer, AT_UINT nSize)
{
        /* Put your memory reallocation code here */
        return( lpBuffer);
};
/*See also example for IG_mem_CB_register() */
```

Remarks:

This callback function is registered by calling IG_mem_CB_register(). The register function must be called prior to any user-defined callback functions being used by the ImageGear library.

1.3.1.3.21 LPFNIG_MPCB_UPDATE

Multi-page images allow you to notify the application about status changes. Use IG_mpi_CB_set to call code that associates the given multi-page image hMIGear with any lpPrivate data, and updates the defined function.

Declaration:

```
typedef  VOID (LPACCUAPI LPFNIG_MPCB_UPDATE)(
   DWORD          dwCBID,
   LPVOID         lpPrivate,
   AT_MODE        nMode,
   UINT           nPage,
   UINT           nCount
);
```

Arguments:

Name	**Type**	**Description**
dwCBID	DWORD	The identifier allocated for this callback function by IG_mpi_CB_set()
lpPrivate	LPVOID	The private data associated with this identifier
nMode	AT_MODE	The type of multi-page image change.
nPage	UINT	Usually the first number of the changed pages.
nCount	UINT	Usually the total number of changed pages.

The following table lists all possible values for nMode and the appropriate sense of the nPage and nCount parameters:

nMode	**nPage**	**nCount**	**Description**
IG_MPCBMODE_MPI_DELETE	Not used	Not used	Notifies the application that the multi-page image is going to be deleted.
IG_MPCBMODE_MPI_ASSOCIATED	Not used	Not used	Notifies the application that the multi-page image is associated with an external file.
IG_MPCBMODE_MPI_CLOSE	No used	Not used	Notifies the application that the multi-page image is going to close the associated external file.
IG_MPCBMODE_MPI_CB_SET	Not used	Not used	Notifies the application that this callback data is set. This notification receives only the callback function that has just been set.
IG_MPCBMODE_MPI_CB_RESET	Not used	Not used	Notifies the application that this callback data is to be reset.
IG_MPCBMODE_MPI_PAGEINSERTED	Index of where new pages start	Number of new pages inserted	Notifies the application that new pages have been inserted into the multi-page image.
IG_MPCBMODE_MPI_PAGEUPDATED	Index of the first updated page	Number of updated pages starting from nPage	Indicates that the application has updated pages in the multi-page image.
IG_MPCBMODE_MPI_PAGEDELETED	Index of first deleted page	Number of deleted pages	Indicates that the application has deleted pages in the multi-page image.
IG_MPCBMODE_MPF_PAGEINSERTED	Index of where new pages start	Number of new pages inserted	Indicates that the application has inserted new pages into the external file image.
IG_MPCBMODE_MPF_PAGEUPDATED	Index of the first updated page	Number of updated pages starting from nPage	Indicates that the application has updated pages in the external multi-page image file.
IG_MPCBMODE_MPF_PAGEDELETED	Index of the first deleted page	Number of deleted pages	Application deleted pages in the external multi-page image file.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This callback function returns a unique identifier, which allows multi-page associations with the given information. After the association is made, the application receives notifications about changes to the multi-page image through this function.

The notification function blocks the execution of the operation that performed the action. This function can be used for thread synchronization. We do not recommended that you call a multi-page API from the notification function (to prevent an unlimited loop from occurring).

1.3.1.3.22 LPFNIG_RASTER_PLANE_SET

This function is called by ImageGear to let your application store or process each color plane raster, as it is obtained from the file.

Declaration:

```
typedef AT_ERRCOUNT (LPACCUAPI LPFNIG_RASTER_PLANE_SET)(
        AT_VOID* lpPrivate,
        const AT_VOID* lpRast,
        AT_PIXPOS cyPos,
        AT_INT cRasterSize,
        AT_INT nBitPlane
);
```

Arguments:

Name	Type	Description
lpPrivate	AT_VOID*	A far pointer to a private data area.
lpRast	const AT_VOID*	Raster line to set.
cyPos	AT_PIXPOS	Y position in the image.
cRasterSize	AT_INT	Size of the raster line.
nBitPlane	AT_INT	Index of the color plane in which to merge.

Return Value:

Returns 0 if successful. Otherwise, returns the number of ImageGear errors that occurred during this function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

ImageGear calls this function to pass raster plane data that has been read from a file to the application. The callback is invoked when reading images where pixel data is stored in planar format. Currently, only TIFF and DICOM format filters support this callback.

1.3.1.3.23 LPFNIG_RASTER_GET

This function is called by ImageGear to obtain from your application each raster to be saved.

Declaration:

```
typedef AT_ERRCOUNT (ACCUAPI LPFNIG_RASTER_GET) (
        LPVOID lpPrivate,
        LPAT_PIXEL lpRast,
        AT_PIXPOS cyPos,
        DWORD cRasterSize
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area.
lpRast	LPAT_PIXEL	Far pointer to first byte of raster your function is providing.
cyPos	AT_PIXPOS	The raster's Y position in the image (0 = top line of image).
dwRasterSize	DWORD	Number of bytes in the raster.

Return Value:

Return an error count.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT  ACCUAPI MyRasterGet (LPVOID lpPrivate, LPAT_PIXEL lpRast, AT_PIXPOS lYpos,
DWORD dwBytes ){
/* Provide raster row lYpos, by storing it where LPAT_PIXEL points.
   Should be exactly dwBytes, counting padding. */
 ...
return IG_error_check();
}
```

Remarks:

This is one of the two types of callback function supplied in calls to IG_save_FD_CB_ex() or IG_save_mem_CB_ex() functions. The line's position in the image is identified by cyPos. The lines may not be in order.

lpRast is a pointer to the start of your raster, and dwRasterSize is the number of bytes in the line. ImageGear will compress the line for you as it saves it, according to the compression scheme specified in your original call to save the image. lpPrivate points to the private data area supplied in that call.

You should check the ImageGear error count (AT_ERRCOUNT) after each raster is read. If you need to terminate the load, you can place an error on the stack yourself, using IG_error_set(). See the description for that function.

1.3.1.3.24 LPFNIG_RASTER_SET

This function is called by ImageGear to provide to your application each raster as it is obtained from the file.

Declaration:

```
typedef AT_ERRCOUNT (ACCUAPI LPFNIG_RASTER_SET) (
        LPVOID lpPrivate,
        const LPAT_PIXEL lpRast,
        AT_PIXPOS cyPos,
        DWORD cRasterSize
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area.
lpRast	const LPAT_PIXEL	Far pointer to first byte of raster that ImageGear is providing on this call.
cyPos	AT_PIXPOS	The raster's Y position in the image (0 = top line of image).
dwRasterSize	DWORD	Number of bytes in the raster.

Return Value:

Return an error count.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT ACCUAPI MyRasterSet (LPVOID lpPrivate, LPAT_PIXEL lpRast, AT_PIXPOS lYpos,
DWORD dwBytes )
{
/* Can use the above information to create a raster or rasters in image bitmap of DIB that
was created by LPFNIG_DIB_CREATE callback.     */
 ...
return IG_error_check();
}
```

Remarks:

This is one of the two types of callback function supplied in calls to IG_load_FD_CB() or IG_load_mem_CB() functions. The line's position in the image is identified by cyPos. The lines may not be in order.

lpRast is a pointer to the start of the raster, and dwRasterSize is the number of bytes in the line. You should not attempt to change the data at lpRast. In general, your application will use this data (along with its knowledge of the width of the image it is creating, any compression scheme, etc.) to create the appropriate raster for the image bitmap of the DIB your application is creating.

See also callback type LPFNIG_DIB_CREATE.

You should check the ImageGear error count (AT_ERRCOUNT) after each raster is read. If you need to terminate the load, you can place an error on the stack yourself, using IG_error_set(). See the description for that function.

1.3.1.3.25 LPFNIG_READ

This function will be called during file operations when a READ is required.

Declaration:

```
typedef LONG (ACCUAPI LPFNIG_READ) (
        LONG fd,
        LPBYTE lpBuffer,
        LONG lSize
);
```

Arguments:

Name	Type	Description
fd	LONG	File Descriptor handle.
lpBuffer	LPBYTE	Far pointer to buffer into which to read data.
lSize	LONG	Number of bytes to read.

Return Value:

Return the number of bytes read, or -1 to indicate that an error occurred.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

See the example for function IG_file_IO_register().

Remarks:

This type of function is established by calling IG_file_IO_register() function.

1.3.1.3.26 LPFNIG_SEEK

This function will be called during file operations when a SEEK is required.

Declaration:

```
typedef AT_INT(ACCUAPI LPFNIG_SEEK) (
        AT_INT fd,
        AT_INT lOffset,
        INT nFlag
);
```

Arguments:

Name	Type	Description
fd	AT_INT	File Descriptor handle.
lOffset	AT_INT	Offset to which to seek.
nFlag	INT	0 = seek from start; 1 = seek from current position; 2 = seek from end.

Return Value:

Return offset into file at the completion of the seek, or -1 to indicate an error occurred.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR         hIGear;       /* HIGEAR handle of image */
LPFNIG_READ    MyReadFunc;    /* To be called for READs */
LPFNIG_WRITE   MyWriteFunc;   /* To be called for WRITEs */
LPFNIG_SEEK    MySeekFunc;    /* To be called for SEEKs  */
{
/* Register Read, Write, and Seek callback functions:   */
IG_file_IO_register ( MyReadFunc, MyWriteFunc, MySeekFunc );
 ...
IG_save_file ( hIGear, "picture.bmp", IG_SAVE_BMP_UNCOMP );
 ...
}
/* This will be called for each seek during the above Save:  */
LONG ACCUAPI  MySeekFunc ( AT_INT fd, AT_INT lOffset, INT nFlag )
{
AT_INT   nResultOffset;
 ...
return  nResultOffset;
}
```

Remarks:

This type of function is established by calling IG_file_IO_register(). This function should return the offset into the file after the seek has completed, or -1 to indicate that an error occurred.

1.3.1.3.27 LPFNIG_SIZE_CHANGE

This function will be called during file operations when a change of file size is required.

Declaration:

```
typedef LONG (LPACCUAPI LPFNIG_SIZE_CHANGE)(
        AT_INT fd,
        AT_INT lSize
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Descriptor ID, from open.
lSize	AT_INT	New size of file, in bytes.

Return Value:

Returns the new size of the file, or -1 to indicate that an error occurred.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This type of function is established by calling IG_file_IO_register().

1.3.1.3.28 LPFNIG_STATUS_BAR

This callback function is called once for each raster (row) in the image.

Declaration:

```
typedef BOOL (ACCUAPI LPFNIG_STATUS_BAR) (
        LPVOID lpPrivate,
        AT_PIXPOS cyPos,
        AT_DIMENSION dwHeight
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area, as specified in your call to IG_status_bar_CB_register() function.
cyPos	AT_PIXPOS	Y position in the image of this raster (row). Calls for the rows are not guaranteed to be in a particular order.
dwHeight	AT_DIMENSION	Total number of rasters (rows) in the image.

Return Value:

Your LPFNIG_STATUS_BAR() callback function should return TRUE if ImageGear should continue the load, save, or print operation it is performing, or FALSE if ImageGear should terminate the operation, placing an IGE_INTERRUPTED_BY_USER error on the error stack.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
BOOL ACCUAPI StatusBar(
   LPVOID            lpPrivate,    /* Private data passed in  */
   AT_PIXPOS            cyPos,     /* Y position in the image */
   AT_DIMENSION      dwHeight      /* Height of the image */
   );
  ...
    /* register the status bar callback function */
    err_count = IG_status_bar_CB_register(StatusBar, &si);
if (err_count == 0)
{
    fSBEnabled = TRUE;
    CheckMenuItem(GetMenu( hWnd),ID_OPTIONS_PROGRESSBAR,MF_CHECKED);
}
                        }
else
                        {
   /* deregister SB function */
   /* deregister the status bar callback function */
   err_count = IG_status_bar_CB_register( NULL, NULL);
if (err_count == 0)
{
   fSBEnabled = FALSE;
   CheckMenuItem(GetMenu( hWnd),ID_OPTIONS_PROGRESSBAR,MF_UNCHECKED);
}
                        }
  ...
```

Remarks:

This is the type of the callback function you specify in calling IG_status_bar_CB_register().

The calls will not necessarily be in row order. See also the description for function IG_status_bar_CB_register().

1.3.1.3.29 LPFNIG_TAG_GET

This function has been deprecated and will be removed from the public API in a future release. Please use LPAFT_IG_METAD_ITEM_ADD_CB instead.

Declaration:

```
typedef BOOL (ACCUAPI LPFNIG_TAG_GET) (
        LPVOID lpPrivate,
        AT_MODE nIGTag,
        LPAT_MODE lpDataType,
        LPVOID lpTagData,
        DWORD dwSize
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	A far pointer to a private data area that can be used for anything you like. For example, you might store the HIGEAR handle of the image that is being loaded.
nIGTag	AT_MODE	ImageGear supplies you with an IGTAG_ Tag ID constant from file Geartags.h, so that you will know which tag is currently being saved.
lpDataType	LPAT_MODE	ImageGear supplies you with an IG_TAG_TYPE_ constant from file accucnst.h, specifying the data type for this tag (e.g., BYTE, LONG, FLOAT, etc.).
lpTagData	LPVOID	ImageGear supplies you with this pointer to a buffer, where you should store your data. If you read this field, you will find that it contains a default value. For example, the ImageGear default value for the Artist tag is "1996-2014 Accusoft Inc., All rights reserved."
dwSize	DWORD	ImageGear tells you the size of tag data buffer, in bytes. This is your limit for the length of what you store to lpTagData.

Return Value:

Returns TRUE if ImageGear should overwrite the default tag data with your data, or FALSE if ImageGear should ignore your data.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
BOOL ACCUAPI TagGet(
   LPVOID         lpPrivate,     /* Private data passed in        */
   AT_MODE        nIGTag,        /* Tag ID from geartags.h        */
   LPAT_MODE      lpType,        /* Type of data lpTag points to  */
   LPBYTE         lpTag,         /* Pointer to tag data           */
   DWORD          dwSize         /* Size of tag data (bytes)      */
   )
...
return TRUE; /* FALSE to terminate the save operation */
}
...
   IG_save_tag_CB_register( TagGet, TagUserGet, (LPVOID)&dwPrivateFlags );
/* The following example may be used for writing additional information into a GIF file:
*/
BOOL ACCUAPI GifCallbackTagGet(
        LPVOID  lpPrivate,
        AT_MODE         nIGTag,
        LPAT_MODE       lpType,
        LPVOID  lpData,
        DWORD   dwSize
        )
{
        LPBYTE  lpRGB;
        static INT i=0;
        switch(nIGTag)
        {
                case IGTAG_GIF_SCREEN_WIDTH:
                        *(LPWORD)lpData=777;
                        break;
                case IGTAG_GIF_SCREEN_FLAGS:
                        *(LPBYTE)lpData=0x80 | 0x07;
                        break;
                case IGTAG_GIF_IMAGE_FLAGS:
                        *(LPBYTE)lpData=0x80 | 0x07;
                        break;
                case IGTAG_GIF_IMAGE_LEFT:
                        *(LPWORD)lpData=111;
                        break;
                case IGTAG_GIF_SCREEN_PALETTE:
                        lpRGB=(LPBYTE)lpData;
                        for(i=0; i<256; i++)
                        {
                                lpRGB[3*i]=(BYTE)(0);
                                lpRGB[3*i+1]=(BYTE)(i);
                                lpRGB[3*i+2]=(BYTE)(0);
                        }
                        break;
                case IGTAG_GIF_IMAGE_PALETTE:
                        lpRGB=(LPBYTE)lpData;
                        for(i=0; i<256; i++)
                        {
                                lpRGB[3*i]=(BYTE)i;
                                lpRGB[3*i+1]=(BYTE)(0);
                                lpRGB[3*i+2]=(BYTE)(0);
                        }
                        break;
                case IGTAG_GIF_EXT_NUMBER_BEFORE_IMG:
                        *(LPWORD)lpData=4;
                        break;
                case IGTAG_GIF_EXT_BEFORE_IMG:
                        switch(i++%4)
                        {
                                case 0:
                                        CtrlExt.bLabel=CTRL_EXT_LABLE;
                                        CtrlExt.bPacked=249;
                                                                {
                                case 0:
                                        CtrlExt.bLabel=CTRL_EXT_LABEL;
                                        CtrlExt.bPacked=249;
                                        CtrlExt.wDelayTime=555;
                                        CtrlExt.bColorIndex=111;
                                        *(LPVOID FAR*)lpData=(LPVOID)&CtrlExt;
                                        break;
                                case 1:
                                        TextExt.bLabel=TEXT_EXT_LABEL;
                                        TextExt.wTextGridLeft=333;
                                        TextExt.lpData=lpStr;
                                        *(LPVOID FAR*)lpData=(LPVOID)&TextExt;
                                        break;
                                case 2:
                                        CommExt.bLabel=COMM_EXT_LABEL;
                                        CommExt.lpData=lpStr;
                                        *(LPVOID FAR*)lpData=(LPVOID)&CommExt;
                                        break;
                                case 3:
                                        ApplExt.bLabel=APPL_EXT_LABEL;
                                        strcpy((CHAR*)ApplExt.Identifier,
                                                "Accusoft");
                                        strcpy((CHAR*)ApplExt.AuthentCode, "6.0");
                                        ApplExt.lpData=lpStr;
                                        *(LPVOID FAR*)lpData=(LPVOID)&ApplExt;
                                        break;
                        }
                        break;
        }
        return TRUE
```

Remarks:

This function will be called once for each "non-volatile" tag that is being written.

This callback function is registered by calling IG_save_tag_CB_register(). When ImageGear is going to perform any save operation it will first check to see if you have registered any applicable callbacks. If you have registered a callback of type LPFNIG_TAG_GET, it will be called once for each "non-volatile" tag that is being written. ImageGear will not call your callback for what it terms "volatile" tags, so that you are unable to modify such tags. If you are writing a TIFF file and want to determine which tags you can write to (that is, which tags are non-volatile), please see the TIFF tags section of Geartags.h. This section is comprised of a list of constants for all registered TIFF tags and a 5-column key that gives you information about the read and write ability of each tag.

See also the Note under TIFF File Format Reference for an explanation of how to use the key.

While you may set the data of a non-volatile tag, your data must not exceed the length specified by dwSize. You must also use the proper data type, which you can check by reading the lpDataType parameter.

Your callback could contain a switch statement for each tag to which you would like to write. Each case in the switch statement could check the ImageGear default setting of the tag and decide whether to change the data. Set your callback to TRUE if you want ImageGear to overwrite the data of a tag, FALSE if you want it to ignore your data. When you set the callback to FALSE, ImageGear will use its own default value for the current tag, but will still call your callback when it parses the next tag.

Note also that some tags can be set only to a certain range of valid values. If your data is out of range for such tags, ImageGear will ignore your data.

If you would like to modify tag data as the image is being read in (for instance, you want it to be modified for display purposes), register a callback of type LPFNIG_TAG_SET.

If you would like to add your own user-defined tags to a TIFF file, register a callback of type LPFNIG_TAG_USER_GET. Your tags will be saved with the image when it is being saved.

See also the discussion about Tag Callbacks in the section Working with ImageGear Callback Functions.

1.3.1.3.30 LPFNIG_TAG_SET

This function has been deprecated and will be removed from the public API in a future release. Please use LPAFT_IG_METAD_ITEM_SET_CB instead.

Declaration:

```
typedef BOOL (ACCUAPI LPFNIG_TAG_SET) (
        LPVOID lpPrivate,
        AT_MODE nIGTag,
        AT_MODE nDataType,
        const LPVOID lpTagData,
        DWORD dwSize
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	A far pointer to a private data area that can be used for anything you like.
nIGTag	AT_MODE	ImageGear supplies you with an IGTAG_ Tag ID constant from file Geartags.h, so that you will know which tag is currently being processed.
nDataType	AT_MODE	ImageGear supplies you with an IG_TAG_TYPE_ constant from file accucnst.h, specifying the data type for this tag (e.g., BYTE, LONG, FLOAT, etc.).
lpTagData	const LPVOID	ImageGear supplies you with a copy of the data from the tag currently being read.
dwSize	DWORD	ImageGear tells you the size of tag data, in bytes.

Return Value:

Returns TRUE if ImageGear should overwrite the default tag data with your data, or FALSE if ImageGear should ignore your data.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
BOOL ACCUAPI TagSet(
   LPVOID         lpPrivate,     /* Private data passed in         */
   AT_MODE        nIGTag,        /* Tag ID from geartags.h         */
   AT_MODE        nType,         /* Type of data in lpTag          */
   const LPVOID   lpTag,         /* Pointer to tag data            */
   DWORD          dwSize         /* Size of tag data (bytes)       */
   )
...
return TRUE; /* FALSE to terminate the operation */
}
...
IG_load_tag_CB_register( TagSet, (LPVOID)&dwPrivateFlags );
...
```

The following example illustrates loading a GIF image.

```
/* Example of LPFNIG_TAG_SET type function, which may be used during loading GIF image:
*/
BOOL ACCUAPI GifCallbackTagSet(
        LPVOID  lpPrivate,
        AT_MODE  nIGTag,
        AT_MODE  nDataType,
        LPVOID  lpData,
        DWORD   dwDataLen
{
        WORD    wWidth;
        WORD    wLeft;
        WORD    wExtNumber;
        LPBYTE                  lpRGB;
        LPAT_GIF_CTRL_EXT lpCtrlExt;
        LPAT_GIF_TEXT_EXT lpTextExt;
        LPAT_GIF_COMM_EXT lpCommExt;
        LPAT_GIF_APPL_EXT lpApplExt;
        LPVOID                  lpExt;
/* Following code illustrates how to get GIF file information
*/
/* Parameter nIGTag in this function informs about kind of data on which points lpData.
*/
/* If GIF image is loaded this parameter may be equal to IGTAG_GIF_ constants, defined in
*/
/* file \ACCUSOFT\GEAR\SOURCE\INCLUDE\geartags.h. For example, when IGTag is equal to*/
/* IGTAG_GIF_SCREEN_ASPECT lpData must point (after conversion) to BYTE - bAspectRatio
field */
/* of GIF_SCREEN_DESC structure. Analogously for other IGTAG_GIF_ constants.*/
/* IGTAG_GIF_SCREEN_BG_COLOR - for getting or setting screen background color index*/
/* - lpData points to BYTE, */
/* 1.When GIF image is loaded: lpData points to GIF Extention Block structure*/
/* 2.When GIF image is written: lpData must points to LPVOID, which points to created by
user*/
/* GIF Extention Block structure. This structure must exist all time when image is
written.
*/
/* IGTAG_GIF_EXT_AFTER_IMG - Analogously as for IGTAG_GIF_EXT_BEFORE_IMG*/
        switch(nIGTag)
        {
                case IGTAG_GIF_SCREEN_WIDTH:
                        wWidth=*(LPWORD)lpData;
                        break;
                case IGTAG_GIF_IMAGE_LEFT:
                        wLeft=*(LPWORD)lpData;
                        break;
                case IGTAG_GIF_SCREEN_PALETTE:
                        lpRGB=(LPBYTE)lpData;
                        break;
                case IGTAG_GIF_IMAGE_PALETTE:
                        lpRGB=(LPBYTE)lpData;
                        break;
                case IGTAG_GIF_EXT_NUMBER_BEFORE_IMG:
                        wExtNumber=*(LPWORD)lpData;
                        break;
                case IGTAG_GIF_EXT_BEFORE_IMG:
                        lpExt=lpData;
                        switch(*(LPDWORD)lpExt)
                        {
                                case CTRL_EXT_LABLE:
                                        lpCtrlExt=(LPAT_GIF_CTRL_EXT)lpExt;
                                        break;
                                case TEXT_EXT_LABLE:
                                        lpTextExt=(LPAT_GIF_TEXT_EXT)lpExt;
                                        break;
                                case COMM_EXT_LABLE:
                                        lpCommExt=(LPAT_GIF_COMM_EXT)lpExt;
                                        break;
                                case APPL_EXT_LABLE:
                                        lpApplExt=(LPAT_GIF_APPL_EXT)lpExt;
                                        break;
                        }
                        break;
                case IGTAG_GIF_EXT_NUMBER_AFTER_IMG:
                        wExtNumber=*(LPWORD)lpData;
                        break;
                case IGTAG_GIF_EXT_AFTER_IMG:
                        lpExt=lpData;
                        switch(*(LPDWORD)lpExt)
                        {
                                case CTRL_EXT_LABLE:
                                        lpCtrlExt=(LPAT_GIF_CTRL_EXT)lpExt;
                                        break;
                                case TEXT_EXT_LABLE:
                                        lpTextExt=(LPAT_GIF_TEXT_EXT)lpExt;
                                        break;
                                case COMM_EXT_LABLE:
                                        lpCommExt=(LPAT_GIF_COMM_EXT)lpExt;
                                        break;
                                case APPL_EXT_LABLE:
                                        lpApplExt=(LPAT_GIF_APPL_EXT)lpExt;
                                        break;
                        }
                        break;
        }
        return TRUE
```

Remarks:

This function will be called once for each tag (both volatile and non-volatile, see LPFNIG_TAG_GET) that is being parsed.

This callback function is registered by calling IG_load_tag_CB_register() function. When ImageGear is going to perform any load operation it will first check to see if you have registered any applicable callbacks. If you have registered a callback of type LPFNIG_TAG_SET, it will be called once for each tag (both volatile and non-volatile, see LPFNIG_TAG_GET) that is being parsed. ImageGear will not let you write data to any tag while loading.

If you would like to modify tag data as the image is being saved, register a callback of type LPFNIG_TAG_GET().

If you would like to add your own user-defined tags to a TIFF file, register a callback of type LPFNIG_USER_TAG_GET. Your tags will be saved with the image when it is being saved.

See also the discussion about Tag Callbacks in the section Working with ImageGear Callback Functions.

The HIGEAR for the image is not created until after all of the tags have been read in.

If you need to store any tag data, make a copy of the data, not the lpTagData pointer.

1.3.1.3.31 LPFNIG_TAG_USER_GET

This function has been deprecated and will be removed from the public API in a future release. Please use LPAFT_IG_METAD_ITEM_GET_CB instead.

Declaration:

```
typedef BOOL (ACCUAPI LPFNIG_TAG_USER_GET) (
        LPVOID lpPrivate,
        LPAT_MODE lpUserTag,
        LPAT_MODE lpDataType,
        LPVOID32 FAR* lpTagData,
        LPDWORD lpSize
);
```

Arguments:

Name	Type	Description
lpPrivate	LPVOID	Far pointer to private data area, which can be used for anything you like, including the storage of the image's HIGEAR handle.
lpIGTag	LPAT_MODE	Far pointer to an AT_MODE variable to receive a private tag number. Your value must be 32768 or higher.
lpDataType	LPAT_MODE	Set to an IG_TAG_TYPE_ constant (these are listed in file accucnst.h) specifying the data type for this tag.
lpTagData	LPVOID32 FAR*	Return a pointer to your data.
lpSize	LPDWORD	Tell ImageGear the size of your data.

Return Value:

Return TRUE if returning a tag, in which case this callback function will be called again in case you have more tags to supply. Return FALSE if you are not returning a tag and are done returning tags.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
BOOL ACCUAPI TagUserGet(
   LPVOID         lpPrivate,     /* Private data passed in */
   LPAT_MODE      lpnIGTag,        /* Tag ID from geartags.h */
   LPAT_MODE      lpType,        /* Type of data lpTag points to */
   LPVOID  FAR*   lpTag,         /* Pointer to tag data */
   LPDWORD        lpSize         /* Size of tag data (bytes) */
   )
/* set Tag ID, data type, length, and pointer to your data */
{
   *lpSize = 0;
   if( (*lpnIGTag>=IGTAG_JPG_APPDATA) && (*lpnIGTag<=IGTAG_JPG_APPDATA_LAST) )
        {
if( (*lpnIGTag-IGTAG_JPG_APPDATA)==12   )        /* supplying only APP13 marker  */
                {
/* supply application marker data */
                        *lpType = IG_TAG_TYPE_RAWBYTES;
                        *lpTag = (LPVOID)(&TestData[0]);
                        *lpSize = sizeof(TestData);
                        return  TRUE;
                }
/* FALSE when no tag being returned */
        }
                      return    FALSE;
}
...
IG_save_tag_CB_register( TagGet, TagUserGet, (LPVOID)&dwPrivateFlags );
...
```

Remarks:

This function allows you to store a private TIFF tag with the image being saved.

This callback function is registered by calling IG_save_tag_CB_register(). In this callback, you supply a tag number, tag type, and tag data. Note that the value of your private tag number must be higher than 32768.

When ImageGear is going to save a file it will check to see if you have registered any applicable callbacks. If you have registered a callback of type LPFNIG_USER_TAG_GET(), it will be called until you set it to FALSE. While your callback is still set to TRUE, it will add your user-defined tags one at a time to the TIFF image being saved.

lpIGTag is a far pointer to an AT_MODE in which you store a privately defined tag ID#. lpDataType is a pointer to your Tag data type, lpTagData is a pointer to your tag data, and lpSize is the length of your tag data.

This callback function should return TRUE when supplying a tag, in which case it will be called again. Return FALSE when you are done supplying tags. This callback will be called at least once if it has been registered.

1.3.1.3.32 LPFNIG_WRITE

This function will be called during file operations when a WRITE is required.

Declaration:

```
typedef LONG (ACCUAPI LPFNIG_WRITE) (
        LONG fd,
        const LPBYTE lpBuffer,
        LONG lSize
);
```

Arguments:

Name	Type	Description
fd	LONG	File Descriptor handle.
lpBuffer	const LPBYTE	Far pointer to buffer from which to write.
lSize	LONG	Number of bytes to write.

Return Value:

Return the number of bytes written, or -1 to indicate that an error occurred.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR       hIGear;        /* HIGEAR handle of image */
LPFNIG_WRITE  MyWriteFunc; /* To be called for file WRITEs */
{
IG_file_IO_register ( NULL, MyWriteFunc, NULL );     /* Register it */
 ...
IG_save_file ( hIGear, "picture.bmp", IG_SAVE_BMP_UNCOMP );
 ...
}
/* This will be called for each write during the above Save:  */
LONG ACCUAPI  MyWriteFunc ( LONG fd, LPBYTE lpBuffer, LONG lNumToWrite )
{
LONG     nNumActuallyWritten;
 ...
return  nNumActuallyWritten; /* Return count, or -1 for error  */
```

Remarks:

This type of function is established by calling IG_file_IO_register().

1.3.1.4 Core Component Structures Reference

This section provides information about the ImageGear Core Component structures, which are organized alphabetically.

- AT_CHANNEL_REF
- AT_COLOR_TEMPERATURE
- AT_DIB
- AT_DIB_EXPORT_OPTIONS
- AT_DPOINT
- AT_DRECTANGLE
- AT_LOGFONT
- AT_POINT
- AT_RECT
- AT_RECTANGLE
- AT_RESOLUTION
- AT_RGB
- AT_RGBQUAD
- AT_ROTATE_MULTIPLE_90_OPTIONS
- AT_SCROLL_INFO
- AT_SRCINFO
- BITMAPINFOHEADER
- tagKERN

1.3.1.4.1 AT_CHANNEL_REF

Defines channel descriptors used for color separation and color combination processing.

```
HIGEAR hImage;
AT_INT uNumber;
```

Members:

Name	**Type**	**Description**
hImage	HIGEAR	Handle to image where channel is being stored.
uNumber	AT_INT	Number of referenced channel in given image.

1.3.1.4.2 AT_COLOR_TEMPERATURE

Defines temperature and tint values used for color temperature processing.

```
AT_DOUBLE Temperature;
AT_DOUBLE Tint;
```

Members:

Name	Type	Description
Temperature	AT_DOUBLE	Temperature value.
Tint	AT_DOUBLE	Tint value.

1.3.1.4.3 AT_DIB

Type of the ImageGear DIB header struct (equivalent to Windows struct BITMAPINFOHEADER).

```
WORD        biSize;
LONG        biWidth;
LONG        biHeight;
WORD        biPlanes;
WORD        biBitCount;
DWORD       biCompression;
DWORD       biSizeImage;
LONG        biXPelsPerMeter;
LONG        biYPelsPerMeter;
DWORD       biClrUsed;
DWORD       biClrImportant;
```

Members:

Name	**Type**	**Description**
biSize	WORD	Total length of this struct, in bytes.
biWidth	LONG	Number of pixels in each raster (row) of the bitmap.
biHeight	LONG	Number of rasters (rows) in the bitmap.
biPlanes	WORD	Number of bit planes. Always = 1.
biBitCount	WORD	Number of bits per pixel (bit depth): 1, 4, 8, 9-16-bit gray level, or 24, 32.
biCompression	DWORD	Type of compression, or 0 (IG_BI_RGB) if bitmap not compressed. IG_BI_RGB, IG_BI_RLE, IG_BI_CMYK.
biSizeImage	DWORD	Total number of bytes in bitmap (necessary if bitmap is compressed; may be 0 if bitmap is IG_BI_RGB).
biXPelsPerMeter	LONG	Pixels per meter horizontally if known (else 0).
biYPelsPerMeter	LONG	Rows per meter vertically if known (else 0).
biClrUsed	DWORD	Number of entries in the color palette that are actually used. (The number of unique pixel values that occur.) If 0, it is assumed all palette entries are used.
biClrImportant	DWORD	Number of palette entries considered important. If 0, all are important.

1.3.1.4.4 AT_DIB_EXPORT_OPTIONS

This structure specifies the options for exporting a DIB from a HIGEAR object.

Declaration:

```
typedef struct tagAT_DIB_EXPORT_OPTIONS
{
   enumIGDIBExportFormats Format;
   AT_RECTANGLE imgRect;
   AT_BOOL UseAlpha;

} AT_DIB_EXPORT_OPTIONS, * LPAT_DIB_EXPORT_OPTIONS;
```

Members:

Name	Type	Description
Format	enumIGDIBExportFormats	Specifies the format for the export.
imgRect	AT_RECTANGLE	Specifies a rectangle area of the image to be exported. If 0 is specified for the width or height, image width or height will be used instead, correspondingly.
UseAlpha	AT_BOOL	Specifies how the Alpha channel shall be handled, if it exists. If TRUE, Alpha channel will be blended into the color channel(s) during the export. If FALSE, Alpha channel will be ignored.

1.3.1.4.5 AT_DPOINT

This struct is simply the X and Y coordinates of a double point.

```
AT_DOUBLE x;
AT_DOUBLE y;
```

Members:

Name	**Type**	**Description**
x	AT_DOUBLE	X coordinate of the double point.
y	AT_DOUBLE	Y coordinate of the double point.

1.3.1.4.6 AT_DRECTANGLE

Stores a set of four integer numbers that represent the location and size of a double rectangle.

```
AT_DOUBLE x;
AT_DOUBLE y;
AT_DOUBLE width;
AT_DOUBLE height;
```

Members:

Name	Type	Description
x	AT_DOUBLE	X coordinate of the upper-left corner of the double rectangle.
y	AT_DOUBLE	Y coordinate of the upper-left corner of the double rectangle.
width	AT_DOUBLE	The width of the double rectangle.
height	AT_DOUBLE	The height of the double rectangle.

1.3.1.4.7 AT_LOGFONT

This structure contains members that specify font in a format similar to MS logical font.

Declaration:

```
typedef struct tagAT_LOGFONT
{
    AT_INT32    lfHeight;
    AT_INT32    lfWidth;
    AT_INT32    lfEscapement;
    AT_INT32    lfOrientation;
    AT_INT32    lfWeight;
    AT_BYTE     lfItalic;
    AT_BYTE     lfUnderline;
    AT_BYTE     lfStrikeOut;
    AT_BYTE     lfCharSet;
    AT_BYTE     lfOutPrecision;
    AT_BYTE     lfClipPrecision;
    AT_BYTE     lfQuality;
    AT_BYTE     lfPitchAndFamily;
    AT_CHAR     lfFaceName[LF_FACESIZE];
} AT_LOGFONT, *LPAT_LOGFONT;
```

Members:

Name	Type	Description
lfHeight	AT_INT32	The height of the font's character cell or character. Expressed in logical units. Also known as the height.
lfWidth	AT_INT32	The average width of characters in the font. Expressed in logical units.
lfEscapement	AT_INT32	The angle between the escapement vector and the x-axis of the device. Expressed in tenths of degrees.
lfOrientation	AT_INT32	The angle between each character’s base line and the x-axis of the device. Expressed in tenths of degrees.
lfWeight	AT_INT32	The weight of the font in the range 0 through 1000. If this value is 0 then the default weight is used.
lfItalic	AT_BYTE	An italic font if set to TRUE.
lfUnderline	AT_BYTE	An underline font if set to TRUE.
lfStrikeout	AT_BYTE	A strikeout font if set to TRUE.
lfCharSet	AT_BYTE	The character set.
lfOutPrecision	AT_BYTE	Defines how closely the output must match the requested font’s height, width, character orientation, escapement, pitch, and font type.
lfClipPrecision	AT_BYTE	Defines how to clip characters that are partially outside the clipping region.
lfQuality	AT_BYTE	Defines how carefully the graphics device interface must attempt to match the logical-font attributes to those of an actual physical font.
lfPitchAndFamily	AT_BYTE	The pitch and family of a font.
lfFaceName	AT_CHAR	A null-terminated string that specifies the font typeface name.

1.3.1.4.8 AT_POINT

This struct is simply the X and Y coordinates of a point.

```
AT_PIXPOS x;
AT_PIXPOS y;
```

Members:

Name	Type	Description
x	AT_PIXPOS	X coordinate of the point.
y	AT_PIXPOS	Y coordinate of the point.

1.3.1.4.9 AT_RECT

This is the type of all rectangles used in calls to ImageGear IG_...() functions (do not confuse with Windows struct type RECT, in which the type of the coordinates is different).

```
AT_PIXPOS left;
AT_PIXPOS top;
AT_PIXPOS right;
AT_PIXPOS bottom;
```

Members:

Name	Type	Description
left	AT_PIXPOS	X coordinate of the upper-left corner of rectangle.
top	AT_PIXPOS	Y coordinate of the upper-left corner of rectangle.
right	AT_PIXPOS	X coordinate of the lower-right corner of rectangle.
bottom	AT_PIXPOS	Y coordinate of the lower-right corner of rectangle.

1.3.1.4.10 AT_RECTANGLE

Stores a set of four integer numbers that represent the location and size of a rectangle.

```
AT_DIMENSION x;
AT_DIMENSION y;
AT_DIMENSION width;
AT_DIMENSION height;
```

Members:

Name	Type	Description
x	AT_DIMENSION	X coordinate of the upper-left corner of the rectangle.
y	AT_DIMENSION	Y coordinate of the upper-left corner of the rectangle.
width	AT_DIMENSION	The width of the rectangle.
height	AT_DIMENSION	The height of the rectangle.

1.3.1.4.11 AT_RESOLUTION

This struct describes an image's resolution. This information is used to map between a number of pixels and a physical length measurement. For example, if you have an image that is 600 pixels wide, and the horizontal resolution is specified as 300 DPI (dots per inch, IG_RESOLUTION_INCHES), then the image should be 2 inches wide when printed.

```
LONG     xResNumerator,
LONG     xResDenominator,
LONG     yResNumerator,
LONG     yResDenominator
AT_LMODE nUnits;
```

Members:

Name	**Type**	**Description**
xResNumerator	LONG	Numerator for horizontal resolution. This number is divided by xResDenominator to determine the horizontal resolution.
xResDenominator	LONG	Denominator for horizontal resolution. Divide this number into xResNumerator to determine the horizontal resolution.
yResNumerator	LONG	Numerator for vertical resolution. This number is divided by yResDenominator to determine the vertical resolution.
yResDenominator	LONG	Denominator for vertical resolution. Divide this number into yResNumerator to determine the vertical resolution.
nUnits	AT_LMODE	Unit type for the resolution. One of the values from enumIGResolutionUnits.

See Also

IG_image_resolution_get()

IG_image_resolution_set()

1.3.1.4.12 AT_RGB

This struct contains three color bytes ordered as in the image bitmap of a standard DIB. Note the order carefully.

```
AT_PIXEL b;
AT_PIXEL g;
AT_PIXEL r;
```

Members:

Name	**Type**	**Description**
b	AT_PIXEL	Byte containing blue component of color, 0 to 255.
g	AT_PIXEL	Green component of color.
r	AT_PIXEL	Red component of color.

1.3.1.4.13 AT_RGBQUAD

Type of a DIB palette entry. This struct contains four bytes; note the order carefully.

```
AT_PIXEL rgbBlue;
AT_PIXEL rgbGreen;
AT_PIXEL rgbRed;
AT_BYTE  rgbReserved;
```

Members:

Name	**Type**	**Description**
rgbBlue	AT_PIXEL	Byte containing blue component of color, 0 to 255.
rgbGreen	AT_PIXEL	Green component of color.
rgbRed	AT_PIXEL	Red component of color.
rgbReserved	AT_BYTE	Reserved, should be 0.

1.3.1.4.14 AT_ROTATE_MULTIPLE_90_OPTIONS

This structure provides rotation options for IG_IP_rotate_multiple_90_opt function.

Declaration:

```
typedef struct AT_ROTATE_MULTIPLE_90_OPTIONS
{
   AT_BOOL SwapResolutions;

} AT_ROTATE_MULTIPLE_90_OPTIONS, * LPAT_ROTATE_MULTIPLE_90_OPTIONS;
```

Members:

Name	**Type**	**Description**
SwapResolutions	AT_BOOL	Specifies whether to swap resolutions when rotating the image by 90 or 270 degrees. If image's horizontal and vertical resolutions are different, setting this field to TRUE preserves its proportions after rotation.

1.3.1.4.15 AT_SCROLL_INFO

Defines display scrolling parameters.

```
AT_INT h_min;
AT_INT h_max;
AT_INT h_cur_pos;
AT_INT h_page;
AT_INT h_line;
AT_INT v_min;
AT_INT v_max;
AT_INT v_cur_pos;
AT_INT v_page;
AT_INT v_line;
```

Members:

Name	**Type**	**Description**
h_min	AT_INT	Minimum horizontal scrolling position.
h_max	AT_INT	Maximum horizontal scrolling position.
h_cur_pos	AT_INT	Current horizontal scrolling position.
h_page	AT_INT	Size of horizontal scroll page.
h_line	AT_INT	Size of horizontal scrolling step.
v_min	AT_INT	Minimum vertical scrolling position.
v_max	AT_INT	Maximum vertical scrolling position.
v_cur_pos	AT_INT	Current vertical scrolling position.
v_page	AT_INT	Size of vertical scroll page.
v_line	AT_INT	Size of vertical scrolling step.

1.3.1.4.16 AT_SRCINFO

Defines the source directory and the formats of the files for batch conversion.

```
AT_CHAR* lpcszSrcDir;
AT_CHAR* lpcszSrcFilter;
```

Members:

Name	**Type**	**Description**
lpcszSrcDir	AT_CHAR*	Source directory for batch conversion.
lpcszSrcFilter	AT_CHAR*	File formats filter to be used for batch conversion.

1.3.1.4.17 BITMAPINFOHEADER

See structure type AT_DIB.

1.3.1.4.18 tagKERN

This structure has been deprecated and will be removed from the public API in a future release.

Declaration:

```
typedef struct tagtagKERN
{
   AT_PIXPOS end_x;
   AT_PIXPOS end_y;
   AT_DIMENSION height;
   AT_INT kern[IG_MAX_KERN_HEIGHT][IG_MAX_KERN_WIDTH];
   AT_DOUBLE normalizer;
   AT_MODE result_form;
   AT_PIXPOS start_x;
   AT_PIXPOS start_y;
   AT_DIMENSION width;

} tagKERN, * LPtagKERN;
```

Structure Members:

Name	Type	Description
end_x	AT_PIXPOS	This field has been deprecated and will be removed from the public API in a future release.
end_y	AT_PIXPOS	This field has been deprecated and will be removed from the public API in a future release.
height	AT_DIMENSION	This field has been deprecated and will be removed from the public API in a future release.
kern	AT_INT	This field has been deprecated and will be removed from the public API in a future release.
normalizer	AT_DOUBLE	This field has been deprecated and will be removed from the public API in a future release.
result_form	AT_MODE	This field has been deprecated and will be removed from the public API in a future release.
start_x	AT_PIXPOS	This field has been deprecated and will be removed from the public API in a future release.
start_y	AT_PIXPOS	This field has been deprecated and will be removed from the public API in a future release.
width	AT_DIMENSION	This field has been deprecated and will be removed from the public API in a future release.

1.3.1.5 Core Component Enumerations Reference

This section provides information about the ImageGear Core Component enumerations, which are organized alphabetically.

- enumAsciiPageSize
- enumBatchCBType
- enumBlendOn
- enumColorProfileAttr
- enumColorProfileGroups
- enumColorProfileStyle
- enumColorSpace
- enumControlNRAOpt
- enumControlOpt
- enumConv24
- enumDIBArea
- enumDIBAreaInfo
- enumDisplayOptions
- enumEncryptModes
- enumEPSFittingMethod
- enumExtention
- enumHTTPVerb
- enumIG_MP_ASSOCIATE
- enumIG_MP_OPENMODE
- enumIG_MPFSaveMode
- enumIG_MPInfoMode
- enumIG_MPISaveMode
- enumIGAlphaChannelType
- enumIGAlphaMode
- enumIGBatchOptions
- enumIGBiCompression
- enumIGBitonalReductModes
- enumIGBlendModes
- enumIGBlurModes
- enumIGBMPTagIDs
- enumIGBTRTagIDs
- enumIGCALTagIDs
- enumIGCIFFCanonCameraSettingsTagIDs
- enumIGCIFFFocalLengthTagIDs
- enumIGCIFFImageInfoTagIDs
- enumIGCIFFPictureInfoTagIDs
- enumIGCIFFShotInfoTagIDs
- enumIGCIFFTagIDs
- enumIGCLPTagIDs
- enumIGColorChannels
- enumIGColorProfileGroups
- enumIGColorSpaceIDs
- enumIGColorSpaces
- enumIGCompressions
- enumIGContrastModes
- enumIGConversionCommands
- enumIGConversionOptions
- enumIGConvolutionResults
- enumIGCursorType
- enumIGCUTTagIDs
- enumIGDCRAWTagIDs
- enumIGDCXTagIDs

- enumIGDepthChangeMode
- enumIGDIBExportFormats
- enumIGDirections
- enumIGDsplAliasModes
- enumIGDsplAlignModes
- enumIGDsplAspectModes
- enumIGDsplBackgroundModes
- enumIGDsplContrastFlags
- enumIGDsplDitheringModes
- enumIGDsplFitModes
- enumIGDsplPaletteModes
- enumIGDsplTranspModes
- enumIGDsplZoomModes
- enumIGEdgeDetectionMethods
- enumIGEdgeMapMethods
- enumIGEPSTagIDs
- enumIGEXIFFPXRTagIDs
- enumIGEXIFGPSTagIDs
- enumIGEXIFInterOperTagIDs
- enumIGEXIFMakerNoteTagIDs
- enumIGEXIFMakerNoteType
- enumIGEXIFTagIDs
- enumIGExtraDataType
- enumIGExtraMode
- enumIGFillOrder
- enumIGFlipModes
- enumIGFltrFormatFlags
- enumIGFormats
- enumIGFrameModes
- enumIGGEMTagIDs
- enumIGGIFTagIDs
- enumIGGrp
- enumIGICATagIDs
- enumIGICDocType
- enumIGICOTagIDs
- enumIGIFFTagIDs
- enumIGIMTTagIDs
- enumIGInterpolations
- enumIGIPTCAppObjAttrTags
- enumIGIPTCAppObjTypeTags
- enumIGIPTCRecord1DatasetTags
- enumIGIPTCRecord2DatasetTags
- enumIGIPTCRecord3DatasetTags
- enumIGIPTCRecord7DatasetTags
- enumIGIPTCRecord8DatasetTags
- enumIGIPTCRecord9DatasetTags
- enumIGIPTCRecordTags
- enumIGIPTCTags
- enumIGJPGTagIDs
- enumIGJPGType
- enumIGKFXTagIDs
- enumIGLicenseType
- enumIGLVTagIDs
- enumIGMergeModes
- enumIGMETADItemType
- enumIGMSPTagIDs

- enumIGMultInfo
- enumIGNCRTagIDs
- enumIGNoiseMethods
- enumIGOrientationModes
- enumIGPaletteFormats
- enumIGPBMTagIDs
- enumIGPCDTagIDs
- enumIGPCXTagIDs
- enumIGPixAccessMode
- enumIGPNGTagIDs
- enumIGPromotionModes
- enumIGPSDTagIDs
- enumIGRASTagIDs
- enumIGResampleInModes
- enumIGResampleOutModes
- enumIGResolutionUnits
- enumIGRotationModes
- enumIGRotationValues
- enumIGSaveFormats
- enumIGSCICTTagIDs
- enumIGSGITagIDs
- enumIGSysDataType
- enumIGTagConstants
- enumIGTags
- enumIGTGATagIDs
- enumIGTIFFTagIDs
- enumIGTwistModes
- enumIGTypeIDs
- enumIGWBMPTagIDs
- enumIGWipeStyles
- enumIGWMFTagIDs
- enumIGWPGTagIDs
- enumIGXBMTagIDs
- enumIGXMPTagIDs
- enumIGXPMTagIDs
- enumIGXWDTagIDs
- enumJPG_DCM
- enumLayoutConstants
- enumLoadColor
- enumLoadDoc
- enumMaxKern
- enumMPAppend
- enumMPCBMODE_MPI
- enumOrientation
- enumPDFSaveFlags
- enumPDFTextEnc
- enumPixdumpComponent
- enumPixdumpComponentEx
- enumPixdumpData
- enumPixdumpMode
- enumPixel
- enumPixelate
- enumPNGCompLevel
- enumPNGStrip
- enumPostScriptLevel
- enumPostScriptType

- enumPrintConstants
- enumRampDirection
- enumRampType
- enumRasterPostProc
- enumRegionIS
- enumROI_IS
- enumScrollTypes
- enumShear
- enumTagTypes
- enumThreadLockMode
- enumTIFFBitonalPaletteMode
- enumTIFFPhoto
- enumTIFFWriteConfig
- enumXWDType

1.3.1.5.1 enumAsciiPageSize

Specifies predefined sizes, in thousandths of an inch, for TXT (ASCII) PAGE_WIDTH, PAGE_HEIGHT filter control parameters.

Values:

IG_ASCIIXSIZELETTER	Letter size width (8 1/2").
IG_ASCIIYSIZELETTER	Letter size height (11").
IG_ASCIIXSIZELEGAL	Legal size width (8 1/2").
IG_ASCIIYSIZELEGAL	Legal size height (14").
IG_ASCIIXSIZEEXECUTIVE	Executive size width (7 1/4").
IG_ASCIIYSIZEEXECUTIVE	Executive size height (10 1/2").
IG_ASCIIXSIZEENVELOPE	Envelope size width (4 1/8").
IG_ASCIIYSIZEENVELOPE	Envelope size height (9 1/2").

1.3.1.5.2 enumBatchCBType

Specifies type of batch I/O callback being registered.

Values:

IG_BATCHCB_BEFORE_OPEN	The callback to call before a file is opened.
IG_BATCHCB_BEFORE_SAVE	The callback to call before a file is saved.

1.3.1.5.3 enumBlendOn

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_BLEND_ON_INTENSITY	This value has been deprecated and will be removed from the public API in a future release.
IG_BLEND_ON_IMAGE	This value has been deprecated and will be removed from the public API in a future release.
IG_BLEND_ON_HUE	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.4 enumColorProfileAttr

GUI color profile window attributes.

Values:

IG_GUI_DATA_COLOR_COMPONENT	Specifies color components to display. Attribute value is a combination of enumGUIColorComponentEx constants.
IG_GUI_DATA_COLORPROFILE_STYLE	Specifies the display style of color components information. Attribute value is a combination of enumColorProfileStyle constants.
IG_GUI_DATA_COLORPROFILE_IMGHWND	Specifies the window handle of GUI window.

Remarks:

See IG_GUI_color_profile_attribute_get for more details.

1.3.1.5.5 enumColorProfileGroups

Identifies a color profile group of the requested color profile.

Values:

IG_CP_GRP_WORKING	WCP (Working Color Profile). This group of color profiles provides information about the default color global parameters used to represent the color data for HIGEAR objects. Those global parameters are used if the image does not have a local color profile associated with it.
IG_CP_GRP_IMPORT	ICP (Import Color Profile). This group of profiles is used during a filter load operation.
IG_CP_GRP_EXPORT	ECP (Export Color Profile). This group of profiles is very similar to ICP but is used in the filter export operation.

1.3.1.5.6 enumColorProfileStyle

GUI color profile window chart attributes.

Values:

IG_GUI_COLORPROFILE_STYLE_STACK	Specifies the chart style. All components data are stacked: every component bar is displayed on top of previous color component bar.
IG_GUI_COLORPROFILE_STYLE_SEPARATE	Specifies the chart style. All components data displayed in separate charts. Each chart has its own base line.
IG_GUI_COLORPROFILE_STYLE_OVERLAID	Specifies the chart style. All components data displayed on single chart with one base line from first color component to the last one.
IG_GUI_COLORPROFILE_STYLE_TRACK	Specifies whether to track mouse pointer movements across image window. Attribute value is TRUE to track mouse or FALSE otherwise.
IG_GUI_COLORPROFILE_STYLE_XY	Specifies whether to display coordinates of start and end mouse positions in GUI window. Attribute value is TRUE to display coordinates or FALSE otherwise.
IG_GUI_COLORPROFILE_STYLE_MASK	Specifies the bit mask used to extract chart style value from GUI window style value.

Remarks:

Specifies the style of data representation and attributes of GUI windows. See IG_GUI_color_profile_attribute_get for more details.

1.3.1.5.7 enumColorSpace

Color space support level.

Values:

IG_CONVERT_TO_RGB	Images are converted to RGB color space during loading. Only affects CMYK images.
IG_FULL_SUPPORT	Full support for loading and saving of image color space.

Remarks:

This enumeration only affects the support of CMYK images. For other color spaces, ImageGear preserves the original image pixel format during loading.

1.3.1.5.8 enumControlNRAOpt

Specifies attributes of non-rectangular ROI associated with an image.

Values:

IG_CONTROL_NR_ROI_DIB	The value specified is a DIB to be used as a mask HIGEAR for the image.
IG_CONTROL_NR_ROI_REFERENCE_POINT	The value specified is a position within the HIGEAR image at which the upper-left corner of the masking HIGEAR should be placed.
IG_CONTROL_NR_ROI_CONDITION	The value specified indicates whether or not ImageGear should override the AT_RECT argument passed to its API. If TRUE then NRA ROI is used, otherwise the rectangular ROI is in effect.
IG_CONTROL_NR_ROI_REFERENCE_POINT_LEFT	The parameter specified is a mask reference point x component.
IG_CONTROL_NR_ROI_REFERENCE_POINT_TOP	The parameter specified is a mask reference point y component.
IG_CONTROL_NR_ROI_VALIDATE	The value returned specifies whether there is a valid mask associated with image.

Remarks:

See IG_IP_NR_ROI_control_set description for more information.

1.3.1.5.9 enumControlOpt

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_CONTROL_JPG_QUALITY	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_DECIMATION_TYPE	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_SAVE_THUMBNAIL	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_THUMBNAIL_WIDTH	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_THUMBNAIL_HEIGHT	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_KEEP_ALPHA	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_TYPE	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_PREDICTOR	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_SCAN_INFO	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_SCAN_INFO_COUNT	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_JPG_LOAD_SCANS	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.

IG_CONTROL_JPG_OLD_LOSSLESS_READ	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_PJPEG_SCAN_INFO	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_PJPEG_SCAN_INFO_COUNT	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_PJPEG_LOAD_SCANS	This value has been deprecated and will be removed from the public API in a future release. See JPEG file format reference for description of JPEG control parameters.
IG_CONTROL_TXT_XDPI	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_YDPI	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_MARGIN_LEFT	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_MARGIN_TOP	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_MARGIN_RIGHT	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_MARGIN_BOTTOM	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_PAGE_WIDTH	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_PAGE_HEIGHT	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_POINT_SIZE	This value has been deprecated and will be removed from

	the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_WEIGHT	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_ITALIC	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_TAB_STOP	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_TYPEFACE	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_LINES_PER_PAGE	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_CHAR_PER_LINE	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_TXT_COMPATIBILITY_MODE	This value has been deprecated and will be removed from the public API in a future release. See TXT file format reference for description of TXT control parameters.
IG_CONTROL_BMP_TYPE	This value has been deprecated and will be removed from the public API in a future release. See BMP file format reference for description of BMP control parameters.
IG_CONTROL_BMP_UPSIDEDOWN	This value has been deprecated and will be removed from the public API in a future release. See BMP file format reference for description of BMP control parameters.
IG_CONTROL_BMP_16GRAY_SCANNER	This value has been deprecated and will be removed from the public API in a future release. See BMP file format reference for description of BMP control parameters.
IG_CONTROL_BMP_16GRAY_SCANNER_TYPE	This value has been deprecated and will be removed from the public API in a future release. See BMP file format reference for description of BMP control parameters.
IG_CONTROL_CCITT_FILL_ORDER	This value has been deprecated and will be removed from the public API in a future release.

	See Group 3 (G3) file format reference for description of G3 control parameters.
IG_CONTROL_CCITT_KFACTOR	This value has been deprecated and will be removed from the public API in a future release. See Group 3 (G3) file format reference for description of G3 control parameters.
IG_CONTROL_TIF_FILENAME_LEN	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_FILENAME	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_FILEDATE_LEN	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_FILEDATE	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_FORCE_SNGL_STRIP	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_BUFFER_SIZE	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_WRITE_FILL_ORDER	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_WRITE_CONFIG	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_PHOTOMETRIC	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_BIGENDIAN	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_DOCUMENTNAME	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control

	parameters.
IG_CONTROL_TIF_DATETIME	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_IMAGE_BEFORE_IFD	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_PLANAR	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_NUMBER_OF_STRIPS	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_WRITE_CLASS_F	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_16_UPDATE_LUT	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_NEW_SUBFILE_TYPE	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_INCLUDE_PAGE_NUMBER	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_IMAGE_WIDTH	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_IMAGE_HEIGHT	This value has been deprecated and will be removed from the public API in a future release. See TIFF file format reference for description of TIF control parameters.
IG_CONTROL_GIF_INTERLACE	This value has been deprecated and will be removed from the public API in a future release. See GIF file format reference for description of GIF control parameters.
IG_CONTROL_GIF_ADD_IMAGE	This value has been deprecated and will be removed from the public API in a future release. See GIF file format reference for description of GIF control parameters.

IG_CONTROL_GIF_VERSION	This value has been deprecated and will be removed from the public API in a future release. See GIF file format reference for description of GIF control parameters.
IG_CONTROL_GIF_EXTBLOCKREADONLY	This value has been deprecated and will be removed from the public API in a future release. See GIF file format reference for description of GIF control parameters.
IG_CONTROL_AVI_FILENAME	This value has been deprecated and will be removed from the public API in a future release. See AVI file format reference for description of AVI control parameters.
IG_CONTROL_KFX_BIT_SEX	This value has been deprecated and will be removed from the public API in a future release. See KFX file format reference for description of KFX control parameters.
IG_CONTROL_PCT_VERSION1	This value has been deprecated and will be removed from the public API in a future release. See PCT file format reference for description of PCT control parameters.
IG_CONTROL_TGA_SAVE_THUMBNAIL	This value has been deprecated and will be removed from the public API in a future release. See TGA file format reference for description of TGA control parameters.
IG_CONTROL_TGA_THUMBNAIL_WIDTH	This value has been deprecated and will be removed from the public API in a future release. See TGA file format reference for description of TGA control parameters.
IG_CONTROL_TGA_THUMBNAIL_HEIGHT	This value has been deprecated and will be removed from the public API in a future release. See TGA file format reference for description of TGA control parameters.
IG_CONTROL_TGA_KEEP_ALPHA	This value has been deprecated and will be removed from the public API in a future release. See TGA file format reference for description of TGA control parameters.
IG_CONTROL_TGA_CONVERT_TO_16	This value has been deprecated and will be removed from the public API in a future release. See TGA file format reference for description of TGA control parameters.
IG_CONTROL_EPS_TIFF_PREVIEW	This value has been deprecated and will be removed from the public API in a future release. See EPS file format reference for description of EPS control parameters.
IG_CONTROL_EPS_FITTING_METHOD	This value has been deprecated and will be removed from the public API in a future release. See EPS file format reference for description of EPS control parameters.
IG_CONTROL_EPS_PIXEL_TO_PIXEL	This value has been deprecated and will be removed from

	the public API in a future release. See EPS file format reference for description of EPS control parameters.
IG_CONTROL_EPS_PAGE_WIDTH	This value has been deprecated and will be removed from the public API in a future release. See EPS file format reference for description of EPS control parameters.
IG_CONTROL_EPS_PAGE_HEIGHT	This value has been deprecated and will be removed from the public API in a future release. See EPS file format reference for description of EPS control parameters.
IG_CONTROL_EPS_XDPI	This value has been deprecated and will be removed from the public API in a future release. See EPS file format reference for description of EPS control parameters.
IG_CONTROL_EPS_YDPI	This value has been deprecated and will be removed from the public API in a future release. See EPS file format reference for description of EPS control parameters.
IG_CONTROL_EPS_TEXTENC	This value has been deprecated and will be removed from the public API in a future release. See EPS file format reference for description of EPS control parameters.
IG_CONTROL_WMF_LOAD_METAFILE	This value has been deprecated and will be removed from the public API in a future release. See WMF file format reference for description of WMF control parameters.
IG_CONTROL_PNG_COMPRESSION	This value has been deprecated and will be removed from the public API in a future release. See PNG file format reference for description of PNG control parameters.
IG_CONTROL_JBIG_STRIP_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_JBIG_TYPICAL_PREDICTOR	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_JBIG_CONTEXT_SHAPE	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_JBIG_TAUX	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_SGI_SAVE_COMPRESSED	This value has been deprecated and will be removed from the public API in a future release. See SGI file format reference for description of SGI control parameters.
IG_CONTROL_PSD_READ_LAYER_MASK	This value has been deprecated and will be removed from the public API in a future release. See PSD file format reference for description of PSD control parameters.
IG_CONTROL_PSB_READ_LAYER_MASK	This value has been deprecated and will be removed from the public API in a future release. See PSB file format reference for description of PSB control

	parameters.
IG_CONTROL_PDF_TEXT_ENCODING	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_PDF_FILENAME	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_PDF_RESOLUTION_X	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_PDF_RESOLUTION_Y	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_PDF_DEPTH	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_PDF_TEXTALPHA	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_PDF_GRAPHICSALPHA	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_PDF_PAGE_WIDTH	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_PDF_PAGE_HEIGHT	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_PDF_INDEPENDENT_PAGESIZE	This value has been deprecated and will be removed from the public API in a future release. See PDF file format reference for description of PDF control parameters.
IG_CONTROL_WLT_QUALITY	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_WL16_QUALITY	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_JPG_ENTROPY_OPTIMIZE	This value has been deprecated and will be removed from the public API in a future release. See JPG file format reference for description of JPG control parameters.

IG_CONTROL_TIF_TILE_H_COUNT	This value has been deprecated and will be removed from the public API in a future release. See TIF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_TILE_V_COUNT	This value has been deprecated and will be removed from the public API in a future release. See TIF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_TILE_WIDTH	This value has been deprecated and will be removed from the public API in a future release. See TIF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_TILE_HEIGHT	This value has been deprecated and will be removed from the public API in a future release. See TIF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_MISSING_COMPRESSION	This value has been deprecated and will be removed from the public API in a future release. See TIF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_WRITE70	This value has been deprecated and will be removed from the public API in a future release. See TIF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_DO_NOT_WRITE_PALETTE	This value has been deprecated and will be removed from the public API in a future release. See TIF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_SAVE_DIFF_PREDICTOR	This value has been deprecated and will be removed from the public API in a future release. See TIF file format reference for description of TIF control parameters.
IG_CONTROL_TIF_SUBIFD_PATH	This value has been deprecated and will be removed from the public API in a future release. See TIF file format reference for description of TIF control parameters.
IG_CONTROL_XWD_TYPE	This value has been deprecated and will be removed from the public API in a future release. See XWD file format reference for description of XWD control parameters.
IG_CONTROL_LURAWAVE_QUALITY	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURAWAVE_SCAN_MODE	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURAWAVE_DWNSCLFACTOR	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_QUANTIZATION	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_TRESHOLD	This value has been deprecated and will be removed from the public API in a future release.

IG_CONTROL_LURADOC_SEGMENTATION	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_TEXTSENSITIVITY	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_RATE	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_BITONAL	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_BACKGROUND	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_QUALITYBACK	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_BACKGROUNDSAMPLE	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_FOREGROUND	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_QUALITYFORE	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_FOREGROUNDSAMPLE	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_THUMBNAIL	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_QUALITYTHUMB	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_THUMBHEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_THUMBWIDTH	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_THUMBSIZE	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURADOC_LAYERS	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_RAW_ALIGNMENT	This value has been deprecated and will be removed from the public API in a future release. See RAW file format reference for description of RAW control parameters.
IG_CONTROL_LURAJP2_RATE_BYTES	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURAJP2_WAVELET_FILTER	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURAJP2_WAVELET_LEVELS	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURAJP2_QUANTIZATION_STYLE	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURAJP2_TILE_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURAJP2_TILE_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_LURAJP2_FILE_FORMAT	This value has been deprecated and will be removed from the public API in a future release.
IG_CONTROL_EXIF_JPEG_SAVE_THUMBNAIL	This value has been deprecated and will be removed from the public API in a future release. See EXIF-JPEG file format reference for description of EXIF-JPEG control parameters.
IG_CONTROL_EXIF_JPEG_THUMBNAIL_WIDTH	This value has been deprecated and will be removed from

	the public API in a future release. See EXIF-JPEG file format reference for description of EXIF-JPEG control parameters.
IG_CONTROL_EXIF_JPEG_THUMBNAIL_HEIGHT	This value has been deprecated and will be removed from the public API in a future release. See EXIF-JPEG file format reference for description of EXIF-JPEG control parameters.
IG_CONTROL_EXIF_JPEG_THUMBNAIL_COMPRESSED	This value has been deprecated and will be removed from the public API in a future release. See EXIF-JPEG file format reference for description of EXIF-JPEG control parameters.
IG_CONTROL_EXIF_JPEG_FLASHPIX_READY	This value has been deprecated and will be removed from the public API in a future release. See EXIF-JPEG file format reference for description of EXIF-JPEG control parameters.
IG_CONTROL_EXIF_TIFF_SAVE_THUMBNAIL	This value has been deprecated and will be removed from the public API in a future release. See EXIF-TIFF file format reference for description of EXIF-TIFF control parameters.
IG_CONTROL_EXIF_TIFF_THUMBNAIL_WIDTH	This value has been deprecated and will be removed from the public API in a future release. See EXIF-TIFF file format reference for description of EXIF-TIFF control parameters.
IG_CONTROL_EXIF_TIFF_THUMBNAIL_HEIGHT	This value has been deprecated and will be removed from the public API in a future release. See EXIF-TIFF file format reference for description of EXIF-TIFF control parameters.

Remarks:

This enumeration has been deprecated and will be removed from the public API in a future release.

Please use IG_fltr_ctrl_get and IG_fltr_ctrl_set functions for accessing filter control parameters. See ImageGear Supported File Formats Reference for a description of individual control parameters.

1.3.1.5.10 enumConv24

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_CONV_24_INTENSITY	This value has been deprecated and will be removed from the public API in a future release.
IG_CONV_24_RGB	This value has been deprecated and will be removed from the public API in a future release.
IG_CONV_24_R	This value has been deprecated and will be removed from the public API in a future release.
IG_CONV_24_G	This value has been deprecated and will be removed from the public API in a future release.
IG_CONV_24_B	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.11 enumDIBArea

Specifies data format to be used by IG_DIB_area_get and IG_DIB_area_set functions.

Values:

IG_DIB_AREA_RAW	This value has been deprecated and will be removed from the public API in a future release.
IG_DIB_AREA_DIB	Get or set the data in standard uncompressed DIB format. Each row is padded to a multiple of 4 bytes length. 1-bit pixels are returned 8 to the byte, most significant bit first. 4-bit pixels are returned 2 to the byte, similarly left justified. 24-bit pixels are returned 3 bytes per pixel, ordered Blue-Green-Red.
IG_DIB_AREA_UNPACKED	Get or set the data using 1 pixel per byte, or 3 bytes for a 24-bit pixel, ordered Blue-Green-Red. Each 1-bit or 4-bit pixel will be returned right justified in a single byte, padded with zeroes in the most significant bits of the byte.

1.3.1.5.12 enumDIBAreaInfo

Specifies modes for IG_IP_area_info_get_ex function.

Values:

IG_DIB_AREA_INFO_MIN	The information requested is a minimum pixel value on the area.
IG_DIB_AREA_INFO_MAX	The information requested is a maximum pixel value on the area.
IG_DIB_AREA_INFO_AVE	The information requested is an average pixel value on the area.
IG_DIB_AREA_INFO_CENTER	The information requested is a value of the central pixel in the area.

1.3.1.5.13 enumDisplayOptions

Specifies display option settings.

Values:

IG_DISPLAY_OPTION_DOWNSHIFT	This option only affects 16-bit grayscale DIBs. Each 16-bit pixel is downshifted by a specified value, and then the least significant word is taken for display. See IG_display_option_set for more details.
IG_DISPLAY_OPTION_LUT	This option only affects 16-bit grayscale DIBs. A lookup table is used to map 16-bit pixels into 8-bit pixel values for display. See IG_display_option_set for more details. If IG_display_option_get is called with IG_DISPLAY_OPTION_LUT parameter when no look up table has been allocated yet, the function allocates a new look up table and returns it to the caller.
IG_PRINT_ADJUST	This value has been deprecated and will be removed from the public API in a future release.
IG_DISPLAY_OPTION_USEMAPMODE	This value has been deprecated and will be removed from the public API in a future release.
IG_DISPLAY_OPTION_DDB_OPTIMIZE	If this parameter is TRUE then monochrome DDB is created from 1bpp HIGEAR image. Otherwise, a compatible bitmap to the current display is created.
IG_DISPLAY_OPTION_OFFSCREEN_DRAW	If this parameter is TRUE then the display code optimizes the drawing of ART/ARTX marks to prevent flashing. Otherwise, each redraw operation is directly displayed, with flashing possible.
IG_DISPLAY_OPTION_OFFSCREEN_WIDTH	This option specifies the width of offscreen drawing surface.
IG_DISPLAY_OPTION_OFFSCREEN_HEIGHT	This option specifies the height of offscreen drawing surface.
IG_DISPLAY_OPTION_LUT_CHECK	This option only affects 16-bit grayscale DIBs, and is only used with IG_display_option_get. IG_display_option_get returns a pointer to the current 16x8 display look up table if it has been set previously, returns NULL otherwise.
IG_DISPLAY_OPTION_LUT8x8	This option only affects 8-bit grayscale DIBs. A lookup table is used to map 8-bit image pixels into 8-bit pixel values for display. See IG_display_option_set for more details. If IG_display_option_get is called with IG_DISPLAY_OPTION_LUT8x8 parameter when no look up table has been allocated yet, the function allocates a new look up table and returns it to the caller.
IG_DISPLAY_OPTION_LUT8x8_CHECK	This option only affects 8-bit grayscale DIBs, and is only used with IG_display_option_get. IG_display_option_get returns a pointer to the current 8x8 display look up table if it has been set previously, returns NULL otherwise.

Remarks:

See IG_display_option_set for more details.

1.3.1.5.14 enumEncryptModes

Specifies different annotation encryption methods. Used by IG_ARTX_encryption_create function.

Values:

IG_ENCRYPT_METHOD_A	Encryption method A.
IG_ENCRYPT_METHOD_B	Encryption method B.
IG_ENCRYPT_METHOD_C	Encryption method C.

1.3.1.5.15 enumEPSFittingMethod

Specifies how to fit the image in the EPS page. Used with EPS filter control parameter FITTING_METHOD.

Values:

IG_EPS_FIT_PAGE	Image is scaled for best fitting the page.
IG_EPS_FIT_ACTUAL	Image is saved with its actual physical dimensions according to image resolution.
IG_EPS_FIT_SET	Image is saved with physical dimensions based on values of X_DPI/Y_DPI control parameters of EPS filter.

1.3.1.5.16 enumExtention

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_EXTENTION_LZW	This value has been deprecated and will be removed from the public API in a future release.
IG_EXTENTION_MEDICAL	This value has been deprecated and will be removed from the public API in a future release.
IG_EXTENTION_ABIC	This value has been deprecated and will be removed from the public API in a future release.
IG_EXTENSION_FLASHPIX	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.17 enumHTTPVerb

Identifies the method of saving image or annotations via the HTTP protocol.

Values:

IG_HTTP_VERB_POST	Used for saving the image or annotation data to file.
IG_HTTP_VERB_PUT	Used for transferring the image or annotation data to the web script.

1.3.1.5.18 enumIG_MP_ASSOCIATE

Specifies association types between a multi-page image and a multi-page image file.

Values:

IG_MP_ASSOCIATE_NONE	There is no association of multi-page image with multi-page image file.
IG_MP_ASSOCIATE_FILE	Multi-page image is associated with multi-page image file.
IG_MP_ASSOCIATE_MEMORY	Reserved for future extensions.

Remarks:

See IG_mpi_info_get for more details.

1.3.1.5.19 enumIG_MP_OPENMODE

Specifies open modes of the associated file of the multi-page image file.

Values:

IG_MP_OPENMODE_NONE	There is no association with multi-page image file.
IG_MP_OPENMODE_READONLY	Associated file has been opened with read-only access mode.
IG_MP_OPENMODE_READWRITE	Associated file has been opened with read-write access mode.

Remarks:

See IG_mpi_file_open for more details.

1.3.1.5.20 enumIG_MPFSaveMode

Specifies file saving modes for IG_mpf_page_save function.

Values:

IG_MPF_SAVE_INSERT	Insert pages into the file at the specified index.
IG_MPF_SAVE_REPLACE	Replace pages starting from the particular page index.

1.3.1.5.21 enumIG_MPInfoMode

Specifies kinds of information returned by IG_mpi_info_get function.

Values:

IG_MP_ASSOCIATION_TYPE	The value returned is the association type of the given multi-page image.
IG_MP_OPEN_MODE	The value returned is the open mode of the associated file.
IG_MP_FILE_NAME	The value returned is the name of the associated file.
IG_MP_MEMBUFFER_PTR	The value returned is the pointer to the memory associated with the multi-page image.
IG_MP_MEMBUFFER_SIZE	The value returned is the size of the associated memory.
IG_MP_FORMAT	The value returned is the file format of the multi-page document. One of the enumIGFormats values.
IG_MP_DOCUMENT	The value returned is the Native document associated with the multi-page document.

1.3.1.5.22 enumIG_MPISaveMode

Specifies file saving modes for IG_mpi_file_save function.

Values:

IG_MPI_SAVE_OVERWRITE	Replace all existing pages with new ones.
IG_MPI_SAVE_APPEND	Append new pages at the end of file.
IG_MPI_SAVE_INSERT	Insert pages into the file at the specified index.
IG_MPI_SAVE_REPLACE	Replace pages starting from the specified page index.

1.3.1.5.23 enumIGAlphaChannelType

This enumeration specifies what bit depth an alpha channel should have.

Values:

IG_ALPHA_CREATE_1	1 Bit.
IG_ALPHA_CREATE_8	8 Bits.

1.3.1.5.24 enumIGAlphaMode

Specifies Alpha channel loading modes.

Values:

IG_ALPHA_MODE_KEEP	Load Alpha channel.
IG_ALPHA_MODE_IGNORE	Ignore Alpha channel.

1.3.1.5.25 enumIGBatchOptions

Identifies the options for batch conversion.

Values:

IG_BATCH_MP_TO_MP	Converts multi-page file to multi-page if the format supports this.
IG_BATCH_RECURSIVE	Recursive conversion.
IG_BATCH_USE_SRC_NAME	Converts file using its src name.

Remarks:

Batch Processing Defines

1.3.1.5.26 enumIGBiCompression

Identifies internal image storage formats used by ImageGear.

Values:

IG_BI_RGB	RGB uncompressed image.
IG_BI_BITFIELDS	This value has been deprecated and will be removed from the public API in a future release.
IG_BI_RLE	RLE compressed 1-bit image.
IG_BI_CMYK	CMYK uncompressed image.
IG_BI_ABIC	This value has been deprecated and will be removed from the public API in a future release.
IG_BI_GRAYSCALE	9-16 bit grayscale uncompressed image.
IG_BI_PSEUDOCOLOR	This value has been deprecated and will be removed from the public API in a future release.
IG_BI_EMPTY	DIB image data has not been allocated.
IG_BI_EXT	This value is used when converting HIGDIBINFO handle to the legacy AT_DIB structure, and specifies that the DIB has some features which AT_DIB does not support.

Remarks:

Please see IG_image_compression_type_get for more information.

1.3.1.5.27 enumIGBitonalReductModes

This enumeration specifies types of bi-tonal color reduction.

Values:

IG_REDUCE_BITONAL_GRAYSCALE	Grayscale.
IG_REDUCE_BITONAL_AVE	AVE.
IG_REDUCE_BITONAL_WEIGHTED	Weighted.

1.3.1.5.28 enumIGBlendModes

This enumeration specifies types of blending of two images.

Values:

IG_BLEND_OVER	Blend the first image over the second image.
IG_BLEND_IN	Blend the first image into the second image.
IG_BLEND_HELD_OUT	The first image is held out by the second image.
IG_BLEND_LINEAR	The first and second images are combined in a linear fashion.

1.3.1.5.29 enumIGBlurModes

This enumeration contains blur kernel sizes.

Values:

IG_BLUR_3	Kernel size 3.
IG_BLUR_5	Kernel size 5.

1.3.1.5.30 enumIGBMPTagIDs

Lists all BMP tag identifiers.

Values:

IGMDTAG_ID_BMP_FORMAT	BMP metadata format identifier.
IGMDTAG_ID_BMP_SIZE	Image size.
IGMDTAG_ID_BMP_WIDTH	Image width.
IGMDTAG_ID_BMP_HEIGHT	Image height.
IGMDTAG_ID_BMP_PLANES	Number of color planes.
IGMDTAG_ID_BMP_BITCOUNT	Image bit count.
IGMDTAG_ID_BMP_COMPRESSION	Image compression type.
IGMDTAG_ID_BMP_XPELSPERMETER	Horizontal resolution in pixels per meter.
IGMDTAG_ID_BMP_YPELSPERMETER	Vertical resolution in pixels per meter.
IGMDTAG_ID_BMP_CLRUSED	Number of color indexes in the color table that are actually used by the bitmap.
IGMDTAG_ID_BMP_CLRIMPORTANT	Number of color indexes required for displaying the bitmap.
IGMDTAG_ID_BMP_UNITS	Type of units used to measure resolution (IBM OS/2 2.x).
IGMDTAG_ID_BMP_RECORDING	Recording algorithm (IBM OS/2 2.x).
IGMDTAG_ID_BMP_RENDERING	Halftoning algorithm used (IBM OS/2 2.x).
IGMDTAG_ID_BMP_SIZE1	Reserved for halftoning algorithm use (IBM OS/2 2.x).
IGMDTAG_ID_BMP_SIZE2	Reserved for halftoning algorithm use (IBM OS/2 2.x).
IGMDTAG_ID_BMP_COLORENCODING	Color model used in bitmap (IBM OS/2 2.x).
IGMDTAG_ID_BMP_IDENTIFIER	Reserved for application use (IBM OS/2 2.x).
IGMDTAG_ID_BMP_TYPE	BMP format type.
IGMDTAG_ID_BMP_REDMASK	Color mask that specifies the red component of each pixel. See BITMAPV4HEADER structure description in the Windows GDI API reference for more details.
IGMDTAG_ID_BMP_GREENMASK	Color mask that specifies the green component of each pixel. See BITMAPV4HEADER structure description in the Windows GDI API reference for more details.
IGMDTAG_ID_BMP_BLUEMASK	Color mask that specifies the blue component of each pixel. See BITMAPV4HEADER structure description in the Windows GDI API reference for more details.
IGMDTAG_ID_BMP_ALPHAMASK	Color mask that specifies the Alpha component of each pixel. See BITMAPV4HEADER structure description in the Windows GDI API reference for more details.
IGMDTAG_ID_BMP_CSTYPE	The color space of the DIB. See BITMAPV4HEADER structure description in the Windows GDI API reference for more details.
IGMDTAG_ID_BMP_ENDPNTCOORDREDX	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_BMP_ENDPNTCOORDREDY	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_BMP_ENDPNTCOORDREDZ	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_BMP_ENDPNTCOORDGREENX	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_BMP_ENDPNTCOORDGREENY	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_BMP_ENDPNTCOORDGREENZ	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_BMP_ENDPNTCOORDBLUEX	This value has been deprecated and will be removed from the public API in a future release.

IGMDTAG_ID_BMP_ENDPNTCOORDBLUEY	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_BMP_ENDPNTCOORDBLUEZ	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_BMP_GAMMARED	Tone response curve for red. See BITMAPV4HEADER structure description in the Windows GDI API reference for more details.
IGMDTAG_ID_BMP_GAMMAGREEN	Tone response curve for green. See BITMAPV4HEADER structure description in the Windows GDI API reference for more details.
IGMDTAG_ID_BMP_GAMMABLUE	Tone response curve for blue. See BITMAPV4HEADER structure description in the Windows GDI API reference for more details.
IGMDTAG_ID_BMP_ENDPNTCOORDS	CIEXYZ coordinates of the red, green and blue endpoints. See BITMAPV4HEADER structure description in the Windows GDI API reference for more details.

1.3.1.5.31 enumIGBTRTagIDs

Lists all BTR tag identifiers.

Values:

IGMDTAG_ID_BTR_FORMAT	BTR metadata format identifier.
IGMDTAG_ID_BTR_MANUFACTURER	Manufacturer value.
IGMDTAG_ID_BTR_VERSION	Version value.
IGMDTAG_ID_BTR_IMAGETYPE	Image type.
IGMDTAG_ID_BTR_HORZRES	Horizontal resolution.
IGMDTAG_ID_BTR_VERTRES	Vertical resolution.
IGMDTAG_ID_BTR_BITSPERPIXEL	Bits per pixel.
IGMDTAG_ID_BTR_PIXELSPERLINE	Pixels per line.
IGMDTAG_ID_BTR_STORAGEFMT	Storage format.
IGMDTAG_ID_BTR_TRANSFMT	Trans format.
IGMDTAG_ID_BTR_PREVPAGE	Previous page.
IGMDTAG_ID_BTR_NEXTPAGE	Next page.
IGMDTAG_ID_BTR_NUMLINES	Number of lines.

1.3.1.5.32 enumIGCALTagIDs

Lists all CAL tag identifiers.

Values:

IGMDTAG_ID_CAL_FORMAT	CAL metadata format identifier.
IGMDTAG_ID_CAL_SPECVERSION	Spec version.
IGMDTAG_ID_CAL_SRCDOCID	Source system document identifier.
IGMDTAG_ID_CAL_DSTDOCID	Destination system document identifier.
IGMDTAG_ID_CAL_TXTFILID	Text file identifier. This record contains a string indicating the document page that this image page contains.
IGMDTAG_ID_CAL_FIGID	Figure or table identifier. This is the number by which the image page figure is referenced.
IGMDTAG_ID_CAL_RTYPE	Raster data type. This is the format of raster image data that follows the header record data block in this file.
IGMDTAG_ID_CAL_RORIENT	Raster image orientation.
IGMDTAG_ID_CAL_RPELCNT	Raster image pel count.
IGMDTAG_ID_CAL_RDENSITY	Raster image density.
IGMDTAG_ID_CAL_SRCGPH	Source system graphics filename.
IGMDTAG_ID_CAL_DOCCLS	Document security label.
IGMDTAG_ID_CAL_FOSIPUBID	PUBLIC identifier of an associated FOSI.
IGMDTAG_ID_CAL_NOTES	Notes information that is not applicable to any of the other records in the CALS raster file header.

1.3.1.5.33 enumIGCIFFCanonCameraSettingsTagIDs

Lists all CIFF Canon Camera Settings tag identifiers.

Values:

IGMDTAG_ID_CIFF_CAMERA_MACRO_MODE	Macro mode.
IGMDTAG_ID_CIFF_CAMERA_SELF_TIMER	Self-timer value.
IGMDTAG_ID_CIFF_CAMERA_QUALITY	Quality setting.
IGMDTAG_ID_CIFF_CAMERA_CANON_FLASH_MODE	Canon flash mode.
IGMDTAG_ID_CIFF_CAMERA_CONTINUOUS_DRIVE	Continuous drive.
IGMDTAG_ID_CIFF_CAMERA_FOCUS_MODE	Focus mode.
IGMDTAG_ID_CIFF_CAMERA_CANON_IMAGE_SIZE	Canon image size.
IGMDTAG_ID_CIFF_CAMERA_EASY_MODE	Easy mode.
IGMDTAG_ID_CIFF_CAMERA_DIGITAL_ZOOM	Digital zoom.
IGMDTAG_ID_CIFF_CAMERA_CONTRAST	Contrast setting.
IGMDTAG_ID_CIFF_CAMERA_SATURATION	Saturation setting.
IGMDTAG_ID_CIFF_CAMERA_SHARPNESS	Sharpness setting.
IGMDTAG_ID_CIFF_CAMERA_ISO	Camera ISO.
IGMDTAG_ID_CIFF_CAMERA_METERING_MODE	Metering mode.
IGMDTAG_ID_CIFF_CAMERA_FOCUS_TYPE	Focus type.
IGMDTAG_ID_CIFF_CAMERA_AFPOINT	AF point setting.
IGMDTAG_ID_CIFF_CAMERA_CANON_EXPOSURE_MODE	Canon exposure mode.
IGMDTAG_ID_CIFF_CAMERA_LENS_TYPE	Lens type.
IGMDTAG_ID_CIFF_CAMERA_LONG_FOCAL	Long focal.
IGMDTAG_ID_CIFF_CAMERA_SHORT_FOCAL	Short focal.
IGMDTAG_ID_CIFF_CAMERA_FOCAL_UNITS	Focal units.
IGMDTAG_ID_CIFF_CAMERA_FLASH_ACTIVITY	Flash activity.
IGMDTAG_ID_CIFF_CAMERA_FLASH_BITS	Flash bits.
IGMDTAG_ID_CIFF_CAMERA_FOCUS_CONTINUOUS	Focus continuous.
IGMDTAG_ID_CIFF_CAMERA_ZOOMED_RESOLUTION	Zoomed resolution.
IGMDTAG_ID_CIFF_CAMERA_ZOOMED_RESOLUTION_BASE	Zoomed resolution base.
IGMDTAG_ID_CIFF_CAMERA_COLOR_TONE	Color tone.

1.3.1.5.34 enumIGCIFFFocalLengthTagIDs

Lists all CIFF Focal Length tag identifiers.

Values:

IGMDTAG_ID_CIFF_FOCAL_LENGTH_LENGTH	Focal length.
IGMDTAG_ID_CIFF_FOCAL_LENGTH_PLANE_XSIZE	Focal plane X size.
IGMDTAG_ID_CIFF_FOCAL_LENGTH_PLANE_YSIZE	Focal plane Y size.

1.3.1.5.35 enumIGCIFFImageInfoTagIDs

Lists all CIFF Image Info tag identifiers.

Values:

IGMDTAG_ID_CIFF_IMAGE_INFO_WIDTH	Width CIFF image Info.
IGMDTAG_ID_CIFF_IMAGE_INFO_HEIGHT	Height CIFF image Info.
IGMDTAG_ID_CIFF_IMAGE_INFO_PIXEL_ASPECT_RATIO	Pixel aspect ratio.
IGMDTAG_ID_CIFF_IMAGE_INFO_ROTATION	Rotation CIFF image Info.
IGMDTAG_ID_CIFF_IMAGE_INFO_COMPONENT_BIT_DEPTH	Component bit depth.
IGMDTAG_ID_CIFF_IMAGE_INFO_COLOR_BIT_DEPTH	Color bit depth.
IGMDTAG_ID_CIFF_IMAGE_INFO_COLOR_BW	Color BW information.

1.3.1.5.36 enumIGCIFFPictureInfoTagIDs

Lists all CIFF Picture Info tag identifiers.

Values:

IGMDTAG_ID_CIFF_PICTURE_IMAGE_WIDTH	Canon image width.
IGMDTAG_ID_CIFF_PICTURE_IMAGE_HEIGHT	Canon image height.
IGMDTAG_ID_CIFF_PICTURE_IMAGE_WIDTH_AS_SHOT	AF image width.
IGMDTAG_ID_CIFF_PICTURE_IMAGE_HEIGHT_AS_SHOT	AF image height.
IGMDTAG_ID_CIFF_PICTURE_AFPOINTS_USED	AF points used.

1.3.1.5.37 enumIGCIFFShotInfoTagIDs

Lists all CIFF Shot Info tag identifiers.

Values:

IGMDTAG_ID_CIFF_SHOT_ISO	Base ISO value.
IGMDTAG_ID_CIFF_SHOT_EXPOSURE_COMPENSATION	Exposure compensation.
IGMDTAG_ID_CIFF_SHOT_WHITE_BALANCE	White balance.
IGMDTAG_ID_CIFF_SHOT_SEQUENCE_NUMBER	Sequence number.
IGMDTAG_ID_CIFF_SHOT_IXUS_AFPOINT	Ixus AF point.
IGMDTAG_ID_CIFF_SHOT_FLASH_EXPOSURE_COMP	Flash exposure comp.
IGMDTAG_ID_CIFF_SHOT_AUTO_EXPOSURE_BRACKETING	Auto exposure bracketing.
IGMDTAG_ID_CIFF_SHOT_AEBBRACKET_VALUE	AEB bracket value.
IGMDTAG_ID_CIFF_SHOT_FOCUS_DISTANCE_UPPER	Focus distance upper.
IGMDTAG_ID_CIFF_SHOT_FOCUS_DISTANCE_LOWER	Focus distance lower.
IGMDTAG_ID_CIFF_SHOT_FNUMBER	F number value.
IGMDTAG_ID_CIFF_SHOT_EXPOSURE_TIME	Exposure time.
IGMDTAG_ID_CIFF_SHOT_BULB_DURATION	Bulb duration.
IGMDTAG_ID_CIFF_SHOT_AUTO_ROTATE	Auto rotate.
IGMDTAG_ID_CIFF_SHOT_SELF_TIMER2	Self-timer 2.

1.3.1.5.38 enumIGCIFFTagIDs

Lists all CIFF tag identifiers.

Values:

IGMDTAG_ID_CIFF_FORMAT	CIFF metadata format identifier.
IGMDTAG_ID_CIFF_NULL_RECORD	Null record.
IGMDTAG_ID_CIFF_FREE_BYTES	Free bytes.
IGMDTAG_ID_CIFF_CANON_COLOR_INFO1	Canon color info 1.
IGMDTAG_ID_CIFF_CANON_FILE_DESCRIPTION	Canon file description.
IGMDTAG_ID_CIFF_USER_COMMENT	User comment.
IGMDTAG_ID_CIFF_CANON_RAW_MAKE_MODEL	Canon raw make model.
IGMDTAG_ID_CIFF_CANON_FIRMWARE_VERSION	Canon firmware version.
IGMDTAG_ID_CIFF_COMPONENT_VERSION	Component version.
IGMDTAG_ID_CIFF_ROM_OPERATION_MODE	ROM operation mode.
IGMDTAG_ID_CIFF_OWNER_NAME	Owner name.
IGMDTAG_ID_CIFF_CANON_IMAGE_TYPE	Canon image type.
IGMDTAG_ID_CIFF_ORIGINAL_FILE_NAME	Original file name.
IGMDTAG_ID_CIFF_THUMBNAIL_FILE_NAME	Thumbnail file name.
IGMDTAG_ID_CIFF_TARGET_IMAGE_TYPE	Target image type.
IGMDTAG_ID_CIFF_SHUTTER_RELEASE_METHOD	Shutter release method.
IGMDTAG_ID_CIFF_SHUTTER_RELEASE_TIMING	Shutter release timing.
IGMDTAG_ID_CIFF_RELEASE_SETTING	Release setting.
IGMDTAG_ID_CIFF_BASE_ISO	Base ISO number.
IGMDTAG_ID_CIFF_FOCAL_LENGTH	Focal length.
IGMDTAG_ID_CIFF_CANON_SHOT_INFO	Canon shot info.
IGMDTAG_ID_CIFF_CANON_COLOR_INFO2	Canon color info 2.
IGMDTAG_ID_CIFF_CANON_CAMERA_SETTINGS	Canon camera settings.
IGMDTAG_ID_CIFF_WHITE_SAMPLE	White sample.
IGMDTAG_ID_CIFF_SENSOR_INFO	Sensor info.
IGMDTAG_ID_CIFF_CANON_CUSTOM_FUNCTIONS	Canon custom functions.
IGMDTAG_ID_CIFF_CANON_PICTURE_INFO	Canon picture info.
IGMDTAG_ID_CIFF_WHITE_BALANCE_TABLE	White balance table.
IGMDTAG_ID_CIFF_COLOR_TEMPERATURE	Color temperature.
IGMDTAG_ID_CIFF_COLOR_SPACE	Color space.
IGMDTAG_ID_CIFF_IMAGE_FORMAT	Image format.
IGMDTAG_ID_CIFF_RECORD_ID	Record ID.
IGMDTAG_ID_CIFF_SELF_TIMER_TIME	Self timer time.
IGMDTAG_ID_CIFF_TARGET_DISTANCE_SETTING	Target distance setting.
IGMDTAG_ID_CIFF_SERIAL_NUMBER	Serial number.
IGMDTAG_ID_CIFF_TIME_STAMP	Time stamp.
IGMDTAG_ID_CIFF_IMAGE_INFO	Image info.
IGMDTAG_ID_CIFF_FLASH_INFO	Flash info.
IGMDTAG_ID_CIFF_MEASURED_EV	Measured EV.
IGMDTAG_ID_CIFF_FILE_NUMBER	File number.
IGMDTAG_ID_CIFF_EXPOSURE_INFO	Exposure info.
IGMDTAG_ID_CIFF_DECODER_TABLE	Decoder table.

IGMDTAG_ID_CIFF_RAW_DATA	The raw data.
IGMDTAG_ID_CIFF_JPG_FROM_RAW	Jpg from raw.
IGMDTAG_ID_CIFF_THUMBNAIL_IMAGE	Thumbnail image.
IGMDTAG_ID_CIFF_IMAGE_DESCRIPTION	Image description.
IGMDTAG_ID_CIFF_CAMERA_OBJECT	Camera object.
IGMDTAG_ID_CIFF_SHOOTING_RECORD	Shooting record.
IGMDTAG_ID_CIFF_MEASURED_INFO	Measured info.
IGMDTAG_ID_CIFF_CAMERA_SPECIFICATION	Camera specification.
IGMDTAG_ID_CIFF_IMAGE_PROPS	Image props.
IGMDTAG_ID_CIFF_EXIF_INFORMATION	Exif information.

1.3.1.5.39 enumIGCLPTagIDs

Lists all CLP tag identifiers.

Values:

IGMDTAG_ID_CLP_FORMAT	CLP metadata format identifier.
IGMDTAG_ID_CLP_FILE_ID	File magic id value.
IGMDTAG_ID_CLP_FORMAT_COUNT	Format count.

1.3.1.5.40 enumIGColorChannels

Specifies color components (channels).

Values:

IG_COLOR_COMP_ALL	All components.
IG_COLOR_COMP_R	R component of RGB color space.
IG_COLOR_COMP_G	G component of RGB color space.
IG_COLOR_COMP_B	B component of RGB color space.
IG_COLOR_COMP_RGB	R, G, B components of RGB color space.
IG_COLOR_COMP_I	I component of indexed RGB color space.
IG_COLOR_COMP_C	C component of CMYK color space.
IG_COLOR_COMP_M	M component of CMYK color space.
IG_COLOR_COMP_Y	Y component of CMYK color space.
IG_COLOR_COMP_K	K component of CMYK color space.
IG_COLOR_COMP_CMYK	C, M, Y, K components of CMYK color space.
IG_COLOR_COMP_YUV_Y	Y component of YUV color space.
IG_COLOR_COMP_YUV_U	U component of YUV color space.
IG_COLOR_COMP_YUV_V	V component of YUV color space.
IG_COLOR_COMP_YUV	Y, U, V components of YUV color space.
IG_COLOR_COMP_LAB_L	L component of LAB color space.
IG_COLOR_COMP_LAB_A	A component of LAB color space.
IG_COLOR_COMP_LAB_B	B component of LAB color space.
IG_COLOR_COMP_LAB	L, A, B components of LAB color space.
IG_COLOR_COMP_IHS_I	I component of IHS color space.
IG_COLOR_COMP_IHS_H	H component of IHS color space.
IG_COLOR_COMP_IHS_S	S component of IHS color space.
IG_COLOR_COMP_IHS	I, H, S components of IHS color space.
IG_COLOR_COMP_HLS_H	H component of HLS color space.
IG_COLOR_COMP_HLS_L	L component of HLS color space.
IG_COLOR_COMP_HLS_S	S component of HLS color space.
IG_COLOR_COMP_HLS	H, L, S component of HLS color space.
IG_COLOR_COMP_HSL_H	This value has been deprecated and will be removed from the public API in a future release. Please use IG_COLOR_COMP_HLS_H instead.
IG_COLOR_COMP_HSL_S	This value has been deprecated and will be removed from the public API in a future release. Please use IG_COLOR_COMP_HLS_S instead.
IG_COLOR_COMP_HSL_L	This value has been deprecated and will be removed from the public API in a future release. Please use IG_COLOR_COMP_HLS_L instead.
IG_COLOR_COMP_HSL	This value has been deprecated and will be removed from the public API in a future release. Please use IG_COLOR_COMP_HLS instead.

1.3.1.5.41 enumIGColorProfileGroups

Identifies a color profile group of the requested color profile.

Values:

IG_CP_GRP_WORKING	WCP (Working Color Profile). This group of color profiles provides information about the default color global parameters used to represent the color data for HIGEAR objects. Those global parameters are used if the image does not have a local color profile associated with it.
IG_CP_GRP_IMPORT	ICP (Import Color Profile). This group of profiles is used during a filter load operation.
IG_CP_GRP_EXPORT	ECP (Export Color Profile). This group of profiles is very similar to ICP but is used in the filter export operation.

1.3.1.5.42 enumIGColorSpaceIDs

Identifies a color space ID. This ID is a bit field which can combine multiple values from enumIGColorSpaceIDs. It can be made up of one, two, or three components. It must describe the color space (RGB, CMYK, grayscale, etc). It may also indicate that a type of alpha channel is present (alpha, pre-multiplied alpha) and/or the presence of one or more extra channels. For example:

IG_COLOR_SPACE_ID_RGB - RGB with no alpha or extra channels.

IG_COLOR_SPACE_ID_Gy Or IG_COLOR_SPACE_ID_A - grayscale with an alpha channel.

IG_COLOR_SPACE_ID_RGB Or IG_COLOR_SPACE_ID_P Or IG_COLOR_SPACE_ID_Ex - RGB with a pre-multiplied alpha channel and one or more extra channels.

Values:

IG_COLOR_SPACE_ID_None	No regular (color) channels, can be combined with alpha and extra values.
IG_COLOR_SPACE_ID_RGB	RGB.
IG_COLOR_SPACE_ID_Gy	Grayscale (intensity).
IG_COLOR_SPACE_ID_I	Indexed RGB.
IG_COLOR_SPACE_ID_IHS	IHS.
IG_COLOR_SPACE_ID_HLS	HLS.
IG_COLOR_SPACE_ID_LAB	LAB.
IG_COLOR_SPACE_ID_YIQ	YIQ.
IG_COLOR_SPACE_ID_CMY	CMY.
IG_COLOR_SPACE_ID_CMYK	CMYK.
IG_COLOR_SPACE_ID_YCbCr	YCbCr.
IG_COLOR_SPACE_ID_YUV	YUV.
IG_COLOR_SPACE_ID_XYZ	For internal use only. CIE XYZ.
IG_COLOR_SPACE_ID_LAST	Equal to last color space (for color channels) enum value.
IG_COLOR_SPACE_ID_ColorMask	Bit mask used to access color space (for color channels) only.
IG_COLOR_SPACE_ID_A	Indicates presence of an alpha channel.
IG_COLOR_SPACE_ID_P	Indicates presence of a pre-multiplied alpha channel.
IG_COLOR_SPACE_ID_Ex	Indicates presence of one or more extra channels.
IG_COLOR_SPACE_ID_RGBA	RGB with alpha channel.
IG_COLOR_SPACE_ID_RGBPA	RGB with pre-multiplied alpha channel.
IG_COLOR_SPACE_ID_GyA	Intensity with alpha channel.
IG_COLOR_SPACE_ID_GyPA	Intensity with pre-multiplied alpha channel.
IG_COLOR_SPACE_ID_RGBAEx	RGB with alpha and extra channels.
IG_COLOR_SPACE_ID_RGBPAEx	RGB with pre-multiplied alpha and extra channels.
IG_COLOR_SPACE_ID_Unknown	Unknown - no color or alpha channels, only extra channels are present.
IG_COLOR_SPACE_ID_HSL	HSL.

1.3.1.5.43 enumIGColorSpaces

Identifies the different color spaces.

Values:

IG_COLOR_SPACE_RGB	RGB.
IG_COLOR_SPACE_I	Intensity.
IG_COLOR_SPACE_IHS	IHS.
IG_COLOR_SPACE_HLS	HLS.
IG_COLOR_SPACE_Lab	Lab.
IG_COLOR_SPACE_YIQ	YIQ.
IG_COLOR_SPACE_CMY	CMY.
IG_COLOR_SPACE_CMYK	CMYK.
IG_COLOR_SPACE_YCrCb	YCrCb.
IG_COLOR_SPACE_YUV	YUV.
IG_COLOR_SPACE_MONO	FlashPix only: 8-bit grayscale.
IG_COLOR_SPACE_ALPHA	FlashPix only: 8-bit alpha.
IG_COLOR_SPACE_MA	FlashPix only: 16-bit: mono + alpha.
IG_COLOR_SPACE_AM	FlashPix only: 16-bit: alpha + mono.
IG_COLOR_SPACE_RGBA	FlashPix only: 32-bit: RGB + alpha.
IG_COLOR_SPACE_ARGB	FlashPix only: 32-bit: alpha + RGB.
IG_COLOR_SPACE_YCC	FlashPix only: 24-bit: photoYCC.
IG_COLOR_SPACE_YCCA	FlashPix only: 32-bit: photoYCC + alpha..
IG_COLOR_SPACE_AYCC	FlashPix only: 32-bit: alpha + photoYCC.
IG_COLOR_SPACE_UNKNOWN	FlashPix only: unknown or invalid color space.
IG_COLOR_SPACE_NOCHANGE	FlashPix only: current color space.
IG_COLOR_SPACE_HSL	HSL.

1.3.1.5.44 enumIGCompressions

Identifies the different format compression schemes.

Values:

IG_COMPRESSION_NONE	No compression.
IG_COMPRESSION_PACKED_BITS	Packed bits compression.
IG_COMPRESSION_HUFFMAN	Huffman encoding.
IG_COMPRESSION_CCITT_G3	CCITT Group 3.
IG_COMPRESSION_CCITT_G4	CCITT Group 4.
IG_COMPRESSION_CCITT_G32D	CCITT Group 3 2D.
IG_COMPRESSION_JPEG	JPEG compression.
IG_COMPRESSION_RLE	Run length encoding.
IG_COMPRESSION_LZW	LZW compression.
IG_COMPRESSION_ABIC_BW	IBM ABIC compression.
IG_COMPRESSION_ABIC_GRAY	IBM ABIC compression.
IG_COMPRESSION_JBIG	IBM JBIG compression.
IG_COMPRESSION_FPX_SINCOLOR	Single color compression.
IG_COMPRESSION_FPX_NOCHANGE	Save with the same compression as loaded.
IG_COMPRESSION_DEFLATE	Deflate compression.
IG_COMPRESSION_IBM_MMR	IBM MMR compression.
IG_COMPRESSION_ABIC	IBM ABIC compression.
IG_COMPRESSION_PROGRESSIVE	Progressive compression (Progressive JPEG and may be PNG in future).
IG_COMPRESSION_EQPC	PowerSDK EQPC(Wavelet) compression.
IG_COMPRESSION_JBIG2	Reserved for future use.
IG_COMPRESSION_LURAWAVE	This value has been deprecated and will be removed from the public API in a future release.
IG_COMPRESSION_LURADOC	This value has been deprecated and will be removed from the public API in a future release.
IG_COMPRESSION_LURAJP2	This value has been deprecated and will be removed from the public API in a future release.
IG_COMPRESSION_ASCII	Image data is converted to ASCII text.
IG_COMPRESSION_RAW	Image data is stored directory in binary raw format.
IG_COMPRESSION_JPEG2K	JPEG2K compression.
IG_COMPRESSION_HDP	HD Photo compression.

1.3.1.5.45 enumIGContrastModes

This enumeration specifies contrast adjustment modes.

Values:

IG_CONTRAST_PALETTE	Alter image palette.
IG_CONTRAST_PIXEL	Alter image pixels.
IG_CONTRAST_AUTO	If the image is indexed, alter palette, otherwise alter pixels.

1.3.1.5.46 enumIGConversionCommands

Identifies the commands for file conversion.

Values:

IG_CONVERT_NONE	Convert with no processing.
IG_CONVERT_ROTATE_90	Convert with rotating to 90 degrees.
IG_CONVERT_ROTATE_180	Convert with rotating to 180 degrees.
IG_CONVERT_ROTATE_270	Convert with rotating to 270 degrees.
IG_CONVERT_FLIP_HORIZONTAL	Convert with flipping horizontal.
IG_CONVERT_FLIP_VERTICAL	Convert with flipping vertical.
IG_CONVERT_TRANSPOSE	Convert and transpose.
IG_CONVERT_TRANSVERSE	Convert and transverse.

1.3.1.5.47 enumIGConversionOptions

Identifies the options for file conversion.

Values:

IG_CONVERT_OPTION_TRIM	Trims image dimensions to a multiple of DCT size.

1.3.1.5.48 enumIGConvolutionResults

This enumeration specifies types of convolution result.

Values:

IG_CONV_RESULT_RAW	The result is stored as is.
IG_CONV_RESULT_ABS	The absolute value of the signed result is stored.
IG_CONV_RESULT_8BIT_SIGNED	The result is stored as 8-bit signed values.
IG_CONV_RESULT_SIGN_CENTERED	The result is stored as 8-bit signed, but 0 is equal to 0x7F. Positive numbers are from 0x7E to 0x00. Negative numbers are from 0x80 to 0xFF. This is used for images that are to be used as background tiles or watermarks.

1.3.1.5.49 enumIGCursorType

These values are used to specify the type of cursor that is displayed under the mouse tracking over the magnifier window.

Values:

IG_GUI_CURSOR_NONE	No cursor.
IG_GUI_CURSOR_APPSTARTING	Standard arrow and small hourglass.
IG_GUI_CURSOR_ARROW	Standard arrow.
IG_GUI_CURSOR_CROSS	Crosshair.
IG_GUI_CURSOR_HAND	Hand (Windows 98/Me, Windows 2000/XP).
IG_GUI_CURSOR_HELP	Arrow and question mark.
IG_GUI_CURSOR_IBEAM	I-beam.
IG_GUI_CURSOR_NO	Slashed circle.
IG_GUI_CURSOR_SIZEALL	Four-pointed arrow pointing North, South, East, and West.
IG_GUI_CURSOR_SIZENESW	Double-pointed arrow pointing North-East and South-West.
IG_GUI_CURSOR_SIZENS	Double-pointed arrow pointing North and South.
IG_GUI_CURSOR_SIZENWSE	Double-pointed arrow pointing North-West and South-East.
IG_GUI_CURSOR_SIZEWE	Double-pointed arrow pointing West and East.
IG_GUI_CURSOR_UPARROW	Vertical arrow.
IG_GUI_CURSOR_WAIT	Hourglass.

1.3.1.5.50 enumIGCUTTagIDs

Lists all CUT tag identifiers.

Values:

IGMDTAG_ID_CUT_FORMAT	CUT metadata format identifier.
IGMDTAG_ID_CUT_WIDTH	Image width.
IGMDTAG_ID_CUT_HEIGHT	Image height.
IGMDTAG_ID_CUT_RESERVED	Reserved value.

1.3.1.5.51 enumIGDCRAWTagIDs

Lists all Digital Camera RAW tag identifiers.

Values:

IGMDTAG_ID_DCRAW_FORMAT	DCRAW metadata format identifier.
IGMDTAG_ID_DCRAW_COMMON	Common metadata section.
IGMDTAG_ID_DCRAW_IMAGEWIDTH	Image width.
IGMDTAG_ID_DCRAW_IMAGEHEIGHT	Image height.
IGMDTAG_ID_DCRAW_BITSPERSAMPLE	Bits per sample.
IGMDTAG_ID_DCRAW_PHOTOMETRICINTERPRETATION	Photometric interpretation.
IGMDTAG_ID_DCRAW_SAMPLESPERPIXEL	Samples per pixel.
IGMDTAG_ID_DCRAW_UNIQUECAMERAMODEL	Unique camera model.
IGMDTAG_ID_DCRAW_MAKE	Camera producer.
IGMDTAG_ID_DCRAW_MODEL	Camera model.
IGMDTAG_ID_DCRAW_TIMESTAMP	Time stamp.
IGMDTAG_ID_DCRAW_CFAREPEATPATTERNDIM	CFA repeat pattern dim.
IGMDTAG_ID_DCRAW_CFAPATTERN	CFA pattern.
IGMDTAG_ID_DCRAW_BLACKLEVELREPEATDIM	Black level repeat dim.
IGMDTAG_ID_DCRAW_BLACKLEVEL	Black level.
IGMDTAG_ID_DCRAW_ASSHOTNEUTRAL	As shot neutral.
IGMDTAG_ID_DCRAW_STATISTICS	Statistics metadata section.
IGMDTAG_ID_DCRAW_WHITELEVEL	White level.
IGMDTAG_ID_DCRAW_CALIBRATIONILLUMINANT1	Calibration illuminant 1.
IGMDTAG_ID_DCRAW_CALIBRATIONILLUMINANT2	Calibration illuminant 2.
IGMDTAG_ID_DCRAW_COLORMATRIX1	Color matrix 1.
IGMDTAG_ID_DCRAW_COLORMATRIX2	Color matrix 2.
IGMDTAG_ID_DCRAW_BASELINEEXPOSURE	Baseline exposure.

1.3.1.5.52 enumIGDCXTagIDs

Lists all DCX tag identifiers.

Values:

IGMDTAG_ID_DCX_FORMAT	DCX metadata format identifier.
IGMDTAG_ID_DCX_MAGIC	Magic value.
IGMDTAG_ID_DCX_PAGE_LIST	Page list.

1.3.1.5.53 enumIGDepthChangeMode

Identifies modes used for changing channel depths of an image.

Values:

IG_DEPTH_CHANGE_NO_SCALE	Changes channel depth without scaling of channel values. This results in unchanged channel values (if the new depth is large enough to accommodate the values). The image will likely change in appearance using this option.
IG_DEPTH_CHANGE_SCALE	Changes channel depth with scaling of channel values. This causes channel values to be scaled so that the ratio of new channel value to new channel depth is as close as possible to the ratio of old channel value to old channel depth. The image will likely maintain the same appearance using this option, unless channel depths are reduced too much to maintain accuracy.

1.3.1.5.54 enumIGDIBExportFormats

Identifies DIB format to be used for a DIB export operation.

Values:

IG_DIB_EXPORT_FORMAT_WINDOWS	Export DIB in standard Windows DIB format. 9-16bpp grayscale images cannot be exported in this format. CMYK images will be exported as 24bpp RGB.
IG_DIB_EXPORT_FORMAT_IG_LEGACY	Export DIB in ImageGear legacy format. In this format, DIB compression can be IG_BI_GRAYSCALE for 9-16bpp grayscale, in which case the DIB bit depth will be the actual bit depth from 9 to 16. Also, DIB compression can be IG_BI_CMYK for CMYK images, in which case the DIB bit depth will be 32.

1.3.1.5.55 enumIGDirections

This enumeration contains general purpose compass directions.

Values:

IG_COMPASS_N	North.
IG_COMPASS_NE	North-East.
IG_COMPASS_E	East.
IG_COMPASS_SE	South-East.
IG_COMPASS_S	South.
IG_COMPASS_SW	South-West.
IG_COMPASS_W	West.
IG_COMPASS_NW	North-West.

1.3.1.5.56 enumIGDsplAliasModes

Identifies image anti-aliasing modes.

IG_DSPL_ANTIALIAS_PRESERVE_BLACK, IG_DSPL_ANTIALIAS_PRESERVE_WHITE, and IG_DSPL_ANTIALIAS_SCALE_TO_GRAY are mutually exclusive.

Values:

IG_DSPL_ANTIALIAS_COLOR	Enables anti-aliasing for downscaled display of non-bi-tonal images.
IG_DSPL_ANTIALIAS_NONE	Anti-aliasing is disabled.
IG_DSPL_ANTIALIAS_PRESERVE_BLACK	Directs ImageGear to preserve black pixels when drawing. Only applicable to downscaled display of bi-tonal images.
IG_DSPL_ANTIALIAS_PRESERVE_WHITE	Directs ImageGear to preserve white pixels when drawing. Only applicable to downscaled display of bi-tonal images.
IG_DSPL_ANTIALIAS_RESAMPLE_BILINE	Enables bi-linear interpolation for upscaled display.
IG_DSPL_ANTIALIAS_SCALE_TO_GRAY	Directs ImageGear to use scale to gray algorithm. Only applicable to downscaled display of bi-tonal images. The image is rendered as 4 bits per pixel grayscale.
IG_DSPL_ANTIALIAS_SUBSAMPLE	Directs ImageGear to use sub-sampling during anti-alias scaling. The output quality is higher and the display speed is considerably faster. Only applicable to downscaled display of bi-tonal images.

1.3.1.5.57 enumIGDsplAlignModes

Identifies the different types of image display alignment modes, i.e., identifies how the displayed image is aligned relative to the device rectangle.

Values:

IG_DSPL_ALIGN_X_LEFT	The image is aligned to the left border of the device rectangle.
IG_DSPL_ALIGN_X_CENTER	The image is centered horizontally.
IG_DSPL_ALIGN_X_RIGHT	The image is aligned to the right border of the device rectangle.
IG_DSPL_ALIGN_Y_LEFT	The image is aligned to the top border of the device rectangle.
IG_DSPL_ALIGN_Y_CENTER	The image is centered vertically.
IG_DSPL_ALIGN_Y_RIGHT	The image is aligned to the bottom border of the device rectangle.

1.3.1.5.58 enumIGDsplAspectModes

Identifies the different types of image's display aspect ratio (i.e., width-to-height ratio).

Values:

IG_DSPL_ASPECT_FIXED	Aspect ratio is the one specified by Aspect Value parameter.
IG_DSPL_ASPECT_NOT_FIXED	Aspect ratio is the one of the device rectangle.

1.3.1.5.59 enumIGDsplBackgroundModes

Identifies the different modes of image background drawing.

Values:

IG_DSPL_BACKGROUND_NONE	The background is disabled, and ImageGear does not fill this area.
IG_DSPL_BACKGROUND_UNDER_IMAGE	The image's transparent pixels are drawn with current background color and background brush. The area outside of Displayed Image Rectangle is not affected.
IG_DSPL_BACKGROUND_BEYOND_IMAGE	The transparent pixels that are outside of Displayed Image Rectangle are drawn with current background color and background brush.
IG_DSPL_BACKGROUND_EVERYWHERE	The background is under the image and beyond the image.

1.3.1.5.60 enumIGDsplContrastFlags

Identifies the different color components of RGB color.

Values:

IG_DSPL_R_CHANNEL	Red color component.
IG_DSPL_G_CHANNEL	Green color component.
IG_DSPL_B_CHANNEL	Blue color component.
IG_DSPL_ALL_CHANNELS	An "OR" combination of 3 flags above; identifies all 3 color components.

1.3.1.5.61 enumIGDsplDitheringModes

Identifies dithering modes and flags.

Values:

IG_DSPL_DITHER_AUTO	Destination device color resolution should be used for dithering. In this mode ImageGear automatically applies dithering only when it is necessary.
IG_DSPL_DITHER_TO_8BPP	Forces ImageGear to assume that the output device is 8 bits per pixel and perform the necessary dithering.
IG_DSPL_DITHER_TO_4BPP	Forces ImageGear to assume that the output device is 4 bits per pixel and perform the necessary dithering.
IG_DSPL_DITHER_TO_1BPP	Forces ImageGear to assume that the output device is 1 bit per pixel and perform the necessary dithering.
IG_DSPL_DITHER_NONE	Disables ImageGear's dithering. In this mode dithering is performed by the operating system or the device driver.
IG_DSPL_DITHER_MODE	Bit mask for dithering modes.
IG_DSPL_DITHER_FIXED_PALETTE	Dithering flag. ImageGear will try to use the standard palette when performing dithering. This may be useful if the output device contains more than one image and by using this flag it is possible to draw images with the same palette.
IG_DSPL_DITHER_NETSCAPE_PALETTE	Dithering flag. Applicable only if the output device is 8 bits per pixel. It directs ImageGear to use the 216 entries of Netscape palette.

1.3.1.5.62 enumIGDsplFitModes

Identifies how an image fits into the device rectangle.

Values:

IG_DSPL_FIT_TO_DEVICE	The image is scaled to fit both the width and height of the device rectangle.
IG_DSPL_FIT_TO_WIDTH	The image is scaled to fit the width of the device rectangle.
IG_DSPL_FIT_TO_HEIGHT	The image is scaled to fit the height of the device rectangle.
IG_DSPL_ACTUAL_SIZE	The device rectangle is ignored, and the image is scaled 1:1.

1.3.1.5.63 enumIGDsplPaletteModes

Identifies the different modes of palette handling.

Values:

IG_DSPL_PALETTE_HIGH	Use the palette in the high priority mode. This means that the operating system palette manager will try to best map colors of DevicePalette to the system palette.
IG_DSPL_PALETTE_LOW	Use the palette in the low priority mode. In this mode the palette manager will try to best preserve current view of the destination while drawing the new image on it.
IG_DSPL_PALETTE_DISABLE	Not to implement DevicePalette in the destination device while drawing the image.

1.3.1.5.64 enumIGDsplTranspModes

Identifies transparency modes.

Values:

IG_DSPL_TRANSPARENCY_NONE	Transparency is disabled.
IG_DSPL_TRANSPARENCY_COLOR	Transparency color is enabled. Pixels which color is equal to the Transparency Color value are displayed transparent when drawing the image.
IG_DSPL_TRANSPARENCY_MASK	Transparency Mask is enabled and the Transparency Mask image is used to specify transparent pixels.
IG_DSPL_TRANSPMASK_STRETCH_TO_IMAGE	If this flag is set, the transparency Mask image is resized and oriented along with the image being displayed. This flag is only used when the transparency mask is enabled.
IG_DSPL_TRANSPMASK_LOCATE_TO_IMAGE	If this flag is set, the transparency mask location is calculated relatively to the Image Rectangle. The mask is oriented along with the image. This flag is only used when the transparency mask is enabled.
IG_DSPL_TRANSPMASK_LOCATE_TO_CLIPRECT	If this flag is set, the transparency mask location is calculated relatively to the Clipping Rectangle. This flag is only used when the transparency mask is enabled.
IG_DSPL_TRANSPMASK_LOCATE_ABSOLUTE	If this flag is set, the transparency mask left-top corner is located according to the MaskLocation option. This flag is only used when the transparency mask is enabled.
IG_DSPL_TRANSPMASK_LOCATE_MODE	Bit mask for mask location modes.

1.3.1.5.65 enumIGDsplZoomModes

Identifies how the image is zoomed in horizontal and vertical directions.

Values:

IG_DSPL_ZOOM_H_MASK	Bit mask for accessing horizontal zoom settings.
IG_DSPL_ZOOM_H_NOT_FIXED	Horizontal zoom factor is not fixed. It is calculated based on other display parameters, such as aspect and fit modes.
IG_DSPL_ZOOM_H_FIXED	Horizontal zoom factor is fixed.
IG_DSPL_ZOOM_V_MASK	Bit mask for accessing vertical zoom settings.
IG_DSPL_ZOOM_V_NOT_FIXED	Vertical zoom factor is not fixed. It is calculated based on other display parameters, such as aspect and fit modes.
IG_DSPL_ZOOM_V_FIXED	Vertical zoom factor is fixed.

1.3.1.5.66 enumIGEdgeDetectionMethods

These constants define the edge detection methods available.

Values:

IG_EDGE_DETECTION_MAXGRADIENT	Edge detection using the maxima of gradient, i.e., maxima of the first order derivative.
IG_EDGE_DETECTION_ZEROXC_DERIV2ND	Edge detection using the zero-crossings of second order derivative along the gradient.
IG_EDGE_DETECTION_DIFF_RECURSIVE	Edge detection using an optimal difference recursive filter.

1.3.1.5.67 enumIGEdgeMapMethods

This enumeration specifies types of edge map operation.

Values:

IG_EDGE_OP_PREWITT	Prewitt.
IG_EDGE_OP_ROBERTS	Roberts.
IG_EDGE_OP_SOBEL	Sobel.
IG_EDGE_OP_LAPLACIAN	Laplacian.
IG_EDGE_OP_LOG	Laplacian of Gaussian.
IG_EDGE_OP_HORIZONTAL	Horizontal.
IG_EDGE_OP_VERTICAL	Vertical.
IG_EDGE_OP_DIAG_POS_45	Diagonal positive.
IG_EDGE_OP_DIAG_NEG_45	Diagonal negative.

1.3.1.5.68 enumIGEPSTagIDs

Lists all EPS tag identifiers.

Values:

IGMDTAG_ID_EPS_FORMAT	EPS metadata format identifier.
IGMDTAG_ID_EPS_VERSION	EPS file version.
IGMDTAG_ID_EPS_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_EPS_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_EPS_TITLE	Document title.
IGMDTAG_ID_EPS_CREATOR	Document creator.
IGMDTAG_ID_EPS_BOUNDINGBOX	Document bounding box.
IGMDTAG_ID_EPS_TRANSLATE	EPS translate.
IGMDTAG_ID_EPS_SCALE	EPS scale.
IGMDTAG_ID_EPS_IMAGE	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.69 enumIGEXIFFPXRTagIDs

Lists all EXIF FPXR tag identifiers.

Values:

IGMDTAG_ID_EXIF_FPX_HEADER	FPXR header.
IGMDTAG_ID_EXIF_FPX_VERSION	FPXR version.
IGMDTAG_ID_EXIF_FPX_EXTENSIONID	FPXR extension ID.
IGMDTAG_ID_EXIF_FPX_INTEROPERABILITYCOUNT	FPXR interoperability count.
IGMDTAG_ID_EXIF_FPX_INDEXTOCONTENTSLIST	FPXR index to contents list.
IGMDTAG_ID_EXIF_FPX_OFFSETTOSTREAM	FPXR offset to stream.
IGMDTAG_ID_EXIF_FPX_STREAMDATA	FPXR stream data.
IGMDTAG_ID_EXIF_FPX_RESERVEDDATA	FPXR reserved data.

1.3.1.5.70 enumIGEXIFGPSTagIDs

Lists all EXIF GPS tag identifiers.

Values:

IGMDTAG_ID_EXIF_GPS_VERSIONID	GPS version ID.
IGMDTAG_ID_EXIF_GPS_LATITUDEREF	GPS latitude ref.
IGMDTAG_ID_EXIF_GPS_LATITUDE	GPS latitude.
IGMDTAG_ID_EXIF_GPS_LONGITUDEREF	GPS longitude ref.
IGMDTAG_ID_EXIF_GPS_LONGITUDE	GPS longitude.
IGMDTAG_ID_EXIF_GPS_ALTITUDEREF	GPS altitude ref.
IGMDTAG_ID_EXIF_GPS_ALTITUDE	GPS altitude.
IGMDTAG_ID_EXIF_GPS_TIMESTAMP	GPS time stamp.
IGMDTAG_ID_EXIF_GPS_SATELLITES	GPS satellites.
IGMDTAG_ID_EXIF_GPS_STATUS	GPS status.
IGMDTAG_ID_EXIF_GPS_MEASUREMODE	GPS measure mode.
IGMDTAG_ID_EXIF_GPS_DOP	Measurement precision.
IGMDTAG_ID_EXIF_GPS_SPEEDREF	GPS speed ref.
IGMDTAG_ID_EXIF_GPS_SPEED	GPS speed.
IGMDTAG_ID_EXIF_GPS_TRACKREF	GPS track ref.
IGMDTAG_ID_EXIF_GPS_TRAK	GPS track.
IGMDTAG_ID_EXIF_GPS_TRACK	GPS track.
IGMDTAG_ID_EXIF_GPS_IMGDIRECTIONREF	GPS img direction ref.
IGMDTAG_ID_EXIF_GPS_IMGDIRECTION	GPS img direction.
IGMDTAG_ID_EXIF_GPS_MAPDATUM	GPS map datum.
IGMDTAG_ID_EXIF_GPS_DESTLATITUDEREF	GPS dest latitude ref.
IGMDTAG_ID_EXIF_GPS_DESTLATITUDE	GPS dest latitude.
IGMDTAG_ID_EXIF_GPS_DESTLONGITUDEREF	GPS dest longitude ref.
IGMDTAG_ID_EXIF_GPS_DESTLONGITUDE	GPS dest longitude.
IGMDTAG_ID_EXIF_GPS_DESTBEARINGREF	GPS dest bearing ref.
IGMDTAG_ID_EXIF_GPS_DESTBEARING	GPS dest bearing.
IGMDTAG_ID_EXIF_GPS_DESTDISTANCEREF	GPS dest distance ref.
IGMDTAG_ID_EXIF_GPS_DESTDISTANCE	GPS dest distance.
IGMDTAG_ID_EXIF_GPS_PROCESSINGMETHOD	GPS processing method.
IGMDTAG_ID_EXIF_GPS_AREAINFORMATION	GPS area information.
IGMDTAG_ID_EXIF_GPS_DATESTAMP	GPS date stamp.
IGMDTAG_ID_EXIF_GPS_DIFFERENTIAL	GPS differential.

1.3.1.5.71 enumIGEXIFInterOperTagIDs

Lists all EXIF Interoperability tag identifiers.

Values:

IGMDTAG_ID_EXIF_IO_INTEROPERABILITYINDEX	Interoperability index.
IGMDTAG_ID_EXIF_IO_INTEROPERABILITYVERSION	Interoperability version.
IGMDTAG_ID_EXIF_IO_RELATEDIMAGEFILEFORMAT	Related image file format.
IGMDTAG_ID_EXIF_IO_RELATEDIMAGEWIDTH	Related image width.
IGMDTAG_ID_EXIF_IO_RELATEDIMAGELENGTH	Related image length.

1.3.1.5.72 enumIGEXIFMakerNoteTagIDs

Lists all general EXIF MakerNote tag identifiers.

Values:

IGMDTAG_ID_EXIF_MAKERNOTE_TYPE	Makernote type.
IGMDTAG_ID_EXIF_MAKERNOTE_PREFIX	Makernote prefix.
IGMDTAG_ID_EXIF_MAKERNOTE_BINARY	Binary data.
IGMDTAG_ID_EXIF_MAKERNOTE_DATA_IFD	Makernote data IFD.

1.3.1.5.73 enumIGEXIFMakerNoteType

Lists all EXIF MakerNote types.

Values:

IG_MAKERNOTE_TYPE_UNKNOWN	Unknown. This is the default type. This type means that ImageGear can't detect Makernote as any other type. Preserving such a makernote and saving it with the file, most likely, does not make any sense, because IFD offsets will be corrupted.
IG_MAKERNOTE_TYPE_IFD	TIFF IFD. Makernote is a valid TIF IFD.
IG_MAKERNOTE_TYPE_IFD_PREFIXED	Prefixed TIFF IFD. Same as IFD, but with a short prefix before the IFD. The prefix is also preserved, so the whole Makernote is preserved when writing to a file.
IG_MAKERNOTE_TYPE_TIF_HEADER_PREFIXED	Makernote starts with a prefix, then goes TIF image header, which points to an IFD.
IG_MAKERNOTE_TYPE_IFD_PREFIXED_OFFSET_II	Makernote starts with a prefix, then goes offset to the IFD, then IFD itself. Makernote IFD uses Intel byte ordering (II), even though the whole file uses Motorola ordering.
IG_MAKERNOTE_TYPES_MAX	Specifies the number of supported makernote types.

1.3.1.5.74 enumIGEXIFTagIDs

Lists all EXIF tag identifiers.

Values:

IGMDTAG_ID_EXIF_JPEG_FORMAT	JPEG metadata format identifier.
IGMDTAG_ID_EXIF_TIFF_FORMAT	TIF metadata format identifier.
IGMDTAG_ID_EXIF_EXPOSURETIME	Exposure time.
IGMDTAG_ID_EXIF_FNUMBER	F number value.
IGMDTAG_ID_EXIF_EXPOSUREPROGRAM	Exposure program.
IGMDTAG_ID_EXIF_SPECTRALSENSITIVITY	Spectral sensitivity.
IGMDTAG_ID_EXIF_ISOSPEEDRATING	ISO speed ratings.
IGMDTAG_ID_EXIF_OECF	Indicates the Opto-Electric Conversion Function (OECF) specified in ISO 14524.
IGMDTAG_ID_EXIF_VERSION	Exif version.
IGMDTAG_ID_EXIF_DATETIMEORIGINAL	Date time original.
IGMDTAG_ID_EXIF_DATETIMEDIGITIZED	Date time digitized.
IGMDTAG_ID_EXIF_COMPONENTCONFIGURATION	Components configuration.
IGMDTAG_ID_EXIF_COMPRESSEDBITSPERPIXEL	Compressed bits per pixel.
IGMDTAG_ID_EXIF_SHUTTERSPEEDVALUE	Shutter speed value.
IGMDTAG_ID_EXIF_APERTUREVALUE	Aperture value.
IGMDTAG_ID_EXIF_BRIGHTNESSVALUE	Brightness value.
IGMDTAG_ID_EXIF_EXPOSUREBIASVALUE	Exposure bias value.
IGMDTAG_ID_EXIF_MAXAPERTUREVALUE	Max aperture value.
IGMDTAG_ID_EXIF_SUBJECTDISTANCE	Subject distance.
IGMDTAG_ID_EXIF_MATERINGMODE	Metering mode.
IGMDTAG_ID_EXIF_METERINGMODE	Metering mode.
IGMDTAG_ID_EXIF_LIGHTSOURCE	Light source.
IGMDTAG_ID_EXIF_FLASH	Indicates whether or not flash used when the image was captured.
IGMDTAG_ID_EXIF_FOCALLENGTH	Focal length.
IGMDTAG_ID_EXIF_SUBJECTAREA	Subject area.
IGMDTAG_ID_EXIF_MAKERNOTE	Maker note.
IGMDTAG_ID_EXIF_USERCOMMENT	User comment.
IGMDTAG_ID_EXIF_SUBSECTIME	Sub sec time.
IGMDTAG_ID_EXIF_SUBSECTIMEORIGINAL	Sub sec time original.
IGMDTAG_ID_EXIF_SUBSECTIMEDIGITIZED	Sub sec time digitized.
IGMDTAG_ID_EXIF_FLASHPIXVERSION	Flash pix version.
IGMDTAG_ID_EXIF_COLORSPACE	Color space.
IGMDTAG_ID_EXIF_PIXELXDIMENSION	Pixel X dimension.
IGMDTAG_ID_EXIF_PIXELYDIMENSION	Pixel Y dimension.
IGMDTAG_ID_EXIF_RELATEDSOUNDFILE	Related sound file.
IGMDTAG_ID_EXIF_INTEROPERABILITYIFD	Interoperability IFD pointer.
IGMDTAG_ID_EXIF_FLASHENERGY	Flash energy.
IGMDTAG_ID_EXIF_SPATIALFREQUENCYRESPONSE	Spatial frequency response.
IGMDTAG_ID_EXIF_FOCALPLANEXRESOLUTION	Focal plane X resolution.
IGMDTAG_ID_EXIF_FOCALPLANEYRESOLUTION	Focal plane Y resolution.
IGMDTAG_ID_EXIF_FOCALPLANERESOLUTIONUNIT	Focal plane resolution unit.

IGMDTAG_ID_EXIF_SUBJECTLOCATION	Subject location.
IGMDTAG_ID_EXIF_EXPOSUREINDEX	Exposure index.
IGMDTAG_ID_EXIF_SENSINGMETHOD	Sensing method.
IGMDTAG_ID_EXIF_FILESOURCE	File source.
IGMDTAG_ID_EXIF_SCENETYPE	Scene type.
IGMDTAG_ID_EXIF_CFAPATTERN	CFA pattern.
IGMDTAG_ID_EXIF_CUSTOMRENDERED	Custom rendered.
IGMDTAG_ID_EXIF_EXPOSUREMODE	Exposure mode.
IGMDTAG_ID_EXIF_WHITEBALANCE	White balance.
IGMDTAG_ID_EXIF_DIGITALZOOMRATIO	Digital zoom ratio.
IGMDTAG_ID_EXIF_FOCALLENGTHIN35MMFILM	Focal length in 35mm film.
IGMDTAG_ID_EXIF_SCENECAPTURETYPE	Scene capture type.
IGMDTAG_ID_EXIF_GAINCONTROL	Gain control.
IGMDTAG_ID_EXIF_CONTRAST	Indicates the direction of contrast processing applied by the camera when the image was shot.
IGMDTAG_ID_EXIF_SATURATION	Indicates the direction of saturation processing applied by the camera when the image was shot.
IGMDTAG_ID_EXIF_SHARPNESS	Indicates the direction of sharpness processing applied by the camera when the image was shot.
IGMDTAG_ID_EXIF_DEVICESETTINGDESCRIPTION	Device setting description.
IGMDTAG_ID_EXIF_SUBJECTDISTANCERANGE	Subject distance range.
IGMDTAG_ID_EXIF_IMAGEUNIQUEID	Indicates an identifier assigned uniquely to each image.
IGMDTAG_ID_EXIF_HEADER	This enumeration value is for internal use only.

1.3.1.5.75 enumIGExtraDataType

Specifies types of vector extra data associated with a HIGEAR image.

Values:

IG_EXTRA_DATA_ARTX	Type of extra data is ARTX.
IG_EXTRA_DATA_CAD	Type of extra data is CAD.
IG_EXTRA_DATA_PDF	Type of extra data is PDF.
IG_EXTRA_DATA_POSTSCRIPT	Type of extra data is PostScript.
IG_EXTRA_DATA_XPS	Type of extra data is XPS.

1.3.1.5.76 enumIGExtraMode

Extra channel loading mode setting.

Values:

IG_EXTRA_MODE_KEEP	Load Extra channels.
IG_EXTRA_MODE_IGNORE	Ignore Extra channels.

1.3.1.5.77 enumIGFillOrder

Identifies the raw bit order.

Values:

IG_FILL_MSB	Little endian bit order.
IG_FILL_LSB	Big endian bit order.

1.3.1.5.78 enumIGFlipModes

This enumeration specifies types of flipping.

Values:

IG_FLIP_HORIZONTAL	Flipping horizontally.
IG_FLIP_VERTICAL	Flipping vertically.

1.3.1.5.79 enumIGFltrFormatFlags

Identifies the format flags such as DETECTSUPPORT, PAGEREADSUPPORT, and other.

Values:

IG_FLTR_DETECTSUPPORT	Format detection is supported.
IG_FLTR_PAGEREADSUPPORT	Page reading is supported.
IG_FLTR_MPAGEREADPSUPPORT	Multi-page reading is supported.
IG_FLTR_MPAGEWRITEPSUPPORT	Multi-page writing is supported.
IG_FLTR_PAGEINSERTSUPPORT	Page insertion is supported.
IG_FLTR_PAGEDELETESUPPORT	Page deleting is supported.
IG_FLTR_PAGESWAPSUPPORT	Page swapping is supported.
IG_FLTR_MPDATASUPPORT	Multi-page data is supported.

1.3.1.5.80 enumIGFormats

Identifies the formats supported by ImageGear.

Values:

IG_FORMAT_ABIC_BILEVEL	IBM ABIC
IG_FORMAT_ABIC_CONCAT	IBM ABIC
IG_FORMAT_AFX	Auto FX
IG_FORMAT_ATT	Not supported
IG_FORMAT_AVI	AVI
IG_FORMAT_BMP	Microsoft Windows Bitmap
IG_FORMAT_BRK	BTR
IG_FORMAT_CAD	Not supported
IG_FORMAT_CAL	CAL
IG_FORMAT_CGM	CGM
IG_FORMAT_CLP	CLP
IG_FORMAT_CUR	Windows Cursors
IG_FORMAT_CUT	CUT
IG_FORMAT_DCM	DICOM
IG_FORMAT_DCRAW	Digital Camera Raw format
IG_FORMAT_DCX	Paintbrush
IG_FORMAT_DGN	DGN
IG_FORMAT_DIB	The same as IG_FORMAT_BMP
IG_FORMAT_DWF	DWF
IG_FORMAT_DWG	DWG
IG_FORMAT_DXF	DXF
IG_FORMAT_EPS	Encapsulated postscript
IG_FORMAT_EXIF_JPEG	Exchangeable image file format
IG_FORMAT_EXIF_TIFF	Exchangeable image file format (EXIF-TIFF)
IG_FORMAT_FPX	FlashPix
IG_FORMAT_G3	Group 3
IG_FORMAT_G32D	Group 3 2D
IG_FORMAT_G4	Group 4
IG_FORMAT_GEM	GEM Raster
IG_FORMAT_GIF	GIF
IG_FORMAT_HLDCRAW	Headerless Digital Camera Raw format
IG_FORMAT_HPGL	HPGL
IG_FORMAT_ICA	IBM IOCA
IG_FORMAT_ICO	Windows icon
IG_FORMAT_IFF	Interchange File Format
IG_FORMAT_IMR	IMR
IG_FORMAT_IMT	IMT
IG_FORMAT_JB2	Reserved for future use.
IG_FORMAT_JBIG	JBIG
IG_FORMAT_JPEG2K	JPEG2000
IG_FORMAT_JPG	JPEG File Interchange
IG_FORMAT_JPX	JPX

IG_FORMAT_KFX	KFX
IG_FORMAT_LURADOC	This value has been deprecated and will be removed from the public API in a future release.
IG_FORMAT_LURAJP2	This value has been deprecated and will be removed from the public API in a future release.
IG_FORMAT_LURAWAVE	This value has been deprecated and will be removed from the public API in a future release.
IG_FORMAT_LV	LV
IG_FORMAT_MAC	MAC
IG_FORMAT_MOD	IBM MO:DCA
IG_FORMAT_MODCA	Not supported
IG_FORMAT_MSP	MSP
IG_FORMAT_MUL	MULTIMEDIA
IG_FORMAT_NCR	NCR
IG_FORMAT_PBM	PBM
IG_FORMAT_PCD	PCD
IG_FORMAT_PCT	Mac PICT
IG_FORMAT_PCX	PC Paintbrush File Format
IG_FORMAT_PDF	Adobe PDF
IG_FORMAT_PGM	Not supported
IG_FORMAT_PJPEG	Not supported
IG_FORMAT_PNG	Portable Network Graphics
IG_FORMAT_PNM	Not supported
IG_FORMAT_POSTSCRIPT	Not supported
IG_FORMAT_PPM	Not supported
IG_FORMAT_PSB	Adobe PSB
IG_FORMAT_PSD	Adobe PSD
IG_FORMAT_PTOCA	PTOCA file
IG_FORMAT_RAS	RAS
IG_FORMAT_RAW	RAW
IG_FORMAT_SCI_CT	Scitex CT file
IG_FORMAT_SCITEX	Not supported
IG_FORMAT_SGI	SGI
IG_FORMAT_STX	Not supported
IG_FORMAT_TGA	TGA
IG_FORMAT_TIF	Tagged Image File Format
IG_FORMAT_TXT	TXT
IG_FORMAT_U3D	U3D format
IG_FORMAT_UNKNOWN	Unknown format
IG_FORMAT_WBMP	Wireless Bitmap File Format
IG_FORMAT_WL16	Not supported
IG_FORMAT_WLT	Not supported
IG_FORMAT_WMF	Windows MetaFile
IG_FORMAT_WPG	WPG
IG_FORMAT_XBM	XBM
IG_FORMAT_XMP	XMP Metadata format
IG_FORMAT_XPM	XPM
IG_FORMAT_XPS	XPS format

IG_FORMAT_XRX	IMG
IG_FORMAT_XWD	XWD

1.3.1.5.81 enumIGFrameModes

This enumeration specifies modes of drawing of a frame.

Values:

IG_DRAW_FRAME_EXPAND	The width and the height of an image are expanded by 2 times the width of the frame.
IG_DRAW_FRAME_OVERWRITE	All four sides of an image are overwritten by the frame.

1.3.1.5.82 enumIGGEMTagIDs

Lists all GEM tag identifiers.

Values:

IGMDTAG_ID_GEM_FORMAT	GEM metadata format identifier.
IGMDTAG_ID_GEM_VERSION	Version valie.
IGMDTAG_ID_GEM_HEADERSIZE	Header size.
IGMDTAG_ID_GEM_PLANES	Color map ID.
IGMDTAG_ID_GEM_PATTERNLENGTH	Pattern length.
IGMDTAG_ID_GEM_WIDTH	Image width.
IGMDTAG_ID_GEM_HEIGHT	Image height.

1.3.1.5.83 enumIGGIFTagIDs

Lists all GIF tag identifiers.

Values:

IGMDTAG_ID_GIF_FORMAT	GIF metadata format identifier.
IGMDTAG_ID_GIF_HEADER	GIF header.
IGMDTAG_ID_GIF_HDR_SIGNATURE	Identifies the GIF Data Stream.
IGMDTAG_ID_GIF_HDR_VERSION	Version number.
IGMDTAG_ID_GIF_SCREEN_DESCRIPTOR	Logical screen descriptor.
IGMDTAG_ID_GIF_SCR_SCREEN_WIDTH	Logical screen width.
IGMDTAG_ID_GIF_SCR_SCREEN_HEIGHT	Logical screen height.
IGMDTAG_ID_GIF_SCR_BACKGROUND_COLOR	Background color index.
IGMDTAG_ID_GIF_SCR_ASPECT_RATIO	Pixel aspect ratio.
IGMDTAG_ID_GIF_SCR_FIELDS	Logical screen packed fields.
IGMDTAG_ID_GIF_SCR_FLD_GL_COLOR_TABLE	Global color table flag.
IGMDTAG_ID_GIF_SCR_FLD_COLOR_RES	Color resolution.
IGMDTAG_ID_GIF_SCR_FLD_SORT	Sort flag.
IGMDTAG_ID_GIF_SCR_FLD_SIZE	Global color table size.
IGMDTAG_ID_GIF_GLOBAL_COLOR_TABLE	Global color table.
IGMDTAG_ID_GIF_IMAGE_DESCRIPTOR	Image descriptor.
IGMDTAG_ID_GIF_IMG_LEFT_POSITION	Image left position.
IGMDTAG_ID_GIF_IMG_TOP_POSITION	Image top position.
IGMDTAG_ID_GIF_IMG_IMAGE_WIDTH	Image width.
IGMDTAG_ID_GIF_IMG_IMAGE_HEIGHT	Image height.
IGMDTAG_ID_GIF_IMG_FIELDS	Image descriptor packed field.
IGMDTAG_ID_GIF_IMG_FLD_LOC_COLOR_TABLE	Local color table flag.
IGMDTAG_ID_GIF_IMG_FLD_INTERLACE	Interlace flag.
IGMDTAG_ID_GIF_IMG_FLD_SORT	Sort flag.
IGMDTAG_ID_GIF_IMG_FLD_SIZE	Local color table size.
IGMDTAG_ID_GIF_LOCAL_COLOR_TABLE	Local color table.
IGMDTAG_ID_GIF_GRAPHIC_CONTROL_EXT	Graphic control extension.
IGMDTAG_ID_GIF_GCE_FIELDS	Graphic control extension packed fields.
IGMDTAG_ID_GIF_GCE_FLD_DISPOSAL_METHOD	Disposal method.
IGMDTAG_ID_GIF_GCE_FLD_USER_INPUT	User input flag.
IGMDTAG_ID_GIF_GCE_FLD_TRANSPARENT	Transparent color flag.
IGMDTAG_ID_GIF_GCE_DELAY_TIME	Delay time.
IGMDTAG_ID_GIF_GCE_TRANSPARENT_COLOR	Transparent color index.
IGMDTAG_ID_GIF_COMMENT_EXTENSION	Comment extension.
IGMDTAG_ID_GIF_PLAIN_TEXT_EXTENSION	Plain text extension.
IGMDTAG_ID_GIF_TXT_GRID_LEFT	Text grid left position.
IGMDTAG_ID_GIF_TXT_GRID_TOP	Text grid top position.
IGMDTAG_ID_GIF_TXT_GRID_WIDTH	Text grid width.
IGMDTAG_ID_GIF_TXT_GRID_HEIGHT	Text grid height.
IGMDTAG_ID_GIF_TXT_CELL_WIDTH	Character cell width.
IGMDTAG_ID_GIF_TXT_CELL_HEIGHT	Character cell height.
IGMDTAG_ID_GIF_TXT_FOREGROUND_COLOR	Text foreground color index.

IGMDTAG_ID_GIF_TXT_BACKGROUND_COLOR	Text background color index.
IGMDTAG_ID_GIF_TXT_TEXT_DATA	Plain text data.
IGMDTAG_ID_GIF_APP_EXTENSION	Application extension.
IGMDTAG_ID_GIF_APP_IDENTIFIER	Application identifier.
IGMDTAG_ID_GIF_APP_AUTH_CODE	Application authentication code.
IGMDTAG_ID_GIF_APP_DATA	Application data.
IGMDTAG_ID_GIF_AFTER_IMAGE_EXT	After image extensions.

1.3.1.5.84 enumIGGrp

Specifies IDs of predefined display parameters groups.

Values:

IG_GRP_DEFAULT	Identifies the group that can be used to display image with default options.
IG_GRP_DEFAULT_PRINT	Identifies the group that can be used to print image with default print options.
IG_GRP_CURRENT_THREAD	Specifies that display group associated with current thread ID should be used for image display.

1.3.1.5.85 enumIGICATagIDs

Lists all ICA tag identifiers.

Values:

IGMDTAG_ID_ICA_FORMAT	IOCA metadata format identifier.
IGMDTAG_ID_ICA_WIDTH	Image width.
IGMDTAG_ID_ICA_HEIGHT	Image height.
IGMDTAG_ID_ICA_DEPTH	Image depth.
IGMDTAG_ID_ICA_XDPI	Horizontal image resolution.
IGMDTAG_ID_ICA_YDPI	Vertical image resolution.
IGMDTAG_ID_ICA_BITORDER	Bit order.
IGMDTAG_ID_ICA_BASE	Size units value.
IGMDTAG_ID_ICA_COMPRESSION	Image compression.
IGMDTAG_ID_ICA_FILLORDER	Fill order.

1.3.1.5.86 enumIGICDocType

This enumeration contains the types of document text or image alignment.

Values:

IG_IC_STANDARD_DOC	Unknown document alignment.
IG_IC_LEFT_ALIGNED_DOC	Left-aligned document with text formed in one column.
IG_IC_RIGHT_ALIGNED_DOC	Right-aligned document with text formed in one column.

1.3.1.5.87 enumIGICOTagIDs

Lists all ICO tag identifiers.

Values:

IGMDTAG_ID_ICO_FORMAT	ICO metadata format identifier.

1.3.1.5.88 enumIGIFFTagIDs

Lists all IFF tag identifiers.

Values:

IGMDTAG_ID_IFF_FORMAT	IFF metadata format identifier.
IGMDTAG_ID_IFF_WIDE	Image width.
IGMDTAG_ID_IFF_HIGH	Image height.
IGMDTAG_ID_IFF_XORG	Image X origin.
IGMDTAG_ID_IFF_YORG	Image Y origin.
IGMDTAG_ID_IFF_PLANES	Color map planes.
IGMDTAG_ID_IFF_MASK	Mask info.
IGMDTAG_ID_IFF_COMPRESSION	Image compression.
IGMDTAG_ID_IFF_TRAN_ASPT	Tran aspt.
IGMDTAG_ID_IFF_PAGE_W	Page width.
IGMDTAG_ID_IFF_PAGE_H	Page height.
IGMDTAG_ID_IFF_VIEW_MODE	View mode.
IGMDTAG_ID_IFF_TRANSP_COLOR	Transp color.
IGMDTAG_ID_IFF_X_ASPECT	X aspect resolution.
IGMDTAG_ID_IFF_Y_ASPECT	Y aspect resolution.

1.3.1.5.89 enumIGIMTTagIDs

Lists all IMT tag identifiers.

Values:

IGMDTAG_ID_IMT_FORMAT	IMT metadata format identifier.
IGMDTAG_ID_IMT_TYPE	IMT file type.
IGMDTAG_ID_IMT_FMT	File format.
IGMDTAG_ID_IMT_HEIGHT	Image height.
IGMDTAG_ID_IMT_WIDTH	Image Width.
IGMDTAG_ID_IMT_RESOLUTION	Image resolution.
IGMDTAG_ID_IMT_BITSWAP	Swap bits.
IGMDTAG_ID_IMT_SWAB	Swap byte.
IGMDTAG_ID_IMT_INVERT	Invert pixel values flag.

1.3.1.5.90 enumIGInterpolations

This enumeration specifies types of interpolation used by ImageGear.

Values:

IG_INTERPOLATION_NONE	No interpolation.
IG_INTERPOLATION_AVERAGE	Average interpolation.
IG_INTERPOLATION_BILINEAR	Bilinear interpolation.
IG_INTERPOLATION_NEAREST_NEIGHBOR	Nearest neighbor interpolation.
IG_INTERPOLATION_PADDING	Resize by adding padding to the image or by cropping the image (this is not an interpolation method).
IG_INTERPOLATION_GRAYSCALE	Scale to Gray interpolation method. It applies to bi-tonal images, and produces a 8-bit grayscale image as a result.
IG_INTERPOLATION_PRESERVE_WHITE	Preserve White interpolation method.
IG_INTERPOLATION_PRESERVE_BLACK	Preserve Black interpolation method.
IG_INTERPOLATION_BICUBIC	Bi-cubic interpolation method.
IG_INTERPOLATION_CANVAS	Same as IG_INTERPOLATION_PADDING.

1.3.1.5.91 enumIGIPTCAppObjAttrTags

Lists IPTC Application Object Attributes.

Values:

IGMDTAG_ID_IPTC_OBJATTR_CURRENT	Object content is about events taking place at the time of the report.
IGMDTAG_ID_IPTC_OBJATTR_ANALYSIS	The object contains data and conclusions drawn by a journalist who has researched the story in depth.
IGMDTAG_ID_IPTC_OBJATTR_ARCHIVE_MATERIAL	The object contains material distributed previously that has been selected from the originator's archives.
IGMDTAG_ID_IPTC_OBJATTR_BACKGROUND	The object provides some scene-setting and explanation for the event being reported.
IGMDTAG_ID_IPTC_OBJATTR_FEATURE	The object content is about a particular event or individual that may not be significant to current breaking news.
IGMDTAG_ID_IPTC_OBJATTR_FORECAST	The object contains opinion as to the outcome of a future event.
IGMDTAG_ID_IPTC_OBJATTR_HISTORY	The object content is based on previous rather than current events.
IGMDTAG_ID_IPTC_OBJATTR_OBITUARY	The object contains a narrative about an individual's life and achievements for publication after his or her death.
IGMDTAG_ID_IPTC_OBJATTR_OPINION	The object contains an editorial comment that reflects the views of the author.
IGMDTAG_ID_IPTC_OBJATTR_POLLS_SURVEYS	The object contains numeric or other information produced as a result of questionnaires or interviews.
IGMDTAG_ID_IPTC_OBJATTR_PROFILE	The object contains a description of the life or activity of a news subject (often a living individual).
IGMDTAG_ID_IPTC_OBJATTR_RES_LISTINGS_TABLES	The object contains alphanumeric data suitable for presentation in tabular form.
IGMDTAG_ID_IPTC_OBJATTR_SIDE_BAR_SUPPORTING_INFO	The object contains a related story that provides additional insight into the news event being reported.
IGMDTAG_ID_IPTC_OBJATTR_SUMMARY	The object is a collection of synopses on news items (generally unrelated).
IGMDTAG_ID_IPTC_OBJATTR_TRANSCRIPT_VERBATIM	The object contains a word-for-word report of a discussion or briefing without significant journalistic intervention.

1.3.1.5.92 enumIGIPTCAppObjTypeTags

Lists IPTC Application Object Types.

Values:

IGMDTAG_ID_IPTC_OBJTYPE_NEWS	Object type is News (default).
IGMDTAG_ID_IPTC_OBJTYPE_DATA	Object type is Data (intended for tables such as statistics or lists, as opposed to narrative text).
IGMDTAG_ID_IPTC_OBJTYPE_ADVISORY	Object type is Advisory (content provider messages, generally not published).

1.3.1.5.93 enumIGIPTCRecord1DatasetTags

Lists all IPTC Record 1 (Envelope) DataSet tags.

Values:

IGMDTAG_ID_IPTC_ENV_MODEL_VERSION	Model version.
IGMDTAG_ID_IPTC_ENV_DESTINATION	Destination information (additional routing information.
IGMDTAG_ID_IPTC_ENV_FILE_FORMAT	File format.
IGMDTAG_ID_IPTC_ENV_FILE_FORMAT_VERSION	File format version.
IGMDTAG_ID_IPTC_ENV_SERVICE_IDENTIFIER	Service identifier.
IGMDTAG_ID_IPTC_ENV_ENVELOPE_NUMBER	Envelope number.
IGMDTAG_ID_IPTC_ENV_PRODUCT_ID	Product ID.
IGMDTAG_ID_IPTC_ENV_ENVELOPE_PRIORITY	Envelope priority.
IGMDTAG_ID_IPTC_ENV_DATE_SENT	Date sent.
IGMDTAG_ID_IPTC_ENV_TIME_SENT	Time sent.
IGMDTAG_ID_IPTC_ENV_CODED_CHARACTER_SET	Coded character set.
IGMDTAG_ID_IPTC_ENV_UNO	Unique Name of Object.
IGMDTAG_ID_IPTC_ENV_ARM_IDENTIFIER	ARM identifier.
IGMDTAG_ID_IPTC_ENV_ARM_VERSION	ARM version.

1.3.1.5.94 enumIGIPTCRecord2DatasetTags

Lists all IPTC Record 2 (Application) DataSet tags.

Values:

IGMDTAG_ID_IPTC_APP_RECORD_VERSION	Record version.
IGMDTAG_ID_IPTC_APP_OBJ_TYPE_REF	Object type reference.
IGMDTAG_ID_IPTC_APP_OBJ_ATTRIBUTE_REF	Object attribute reference.
IGMDTAG_ID_IPTC_APP_OBJ_NAME	Object name.
IGMDTAG_ID_IPTC_APP_EDIT_STATUS	Edit status.
IGMDTAG_ID_IPTC_APP_EDITORIAL_UPDATE	Editorial update.
IGMDTAG_ID_IPTC_APP_URGENCY	Editorial urgency.
IGMDTAG_ID_IPTC_APP_SUBJECT_REFERENCE	Subject reference.
IGMDTAG_ID_IPTC_APP_CATEGORY	Category that identifies the subject of the object in the opinion of the image provider.
IGMDTAG_ID_IPTC_APP_SUPPLEMENTAL_CATEGORY	Supplemental category.
IGMDTAG_ID_IPTC_APP_FIXTURE_IDENTIFIER	Fixture identifier.
IGMDTAG_ID_IPTC_APP_KEYWORDS	Keywords to description.
IGMDTAG_ID_IPTC_APP_CONTENT_LOCATION_CODE	Content location code.
IGMDTAG_ID_IPTC_APP_CONTENT_LOCATION_NAME	Content location name.
IGMDTAG_ID_IPTC_APP_RELEASE_DATE	Release date.
IGMDTAG_ID_IPTC_APP_RELEASE_TIME	Release time.
IGMDTAG_ID_IPTC_APP_EXPIRATION_DATE	Expiration date.
IGMDTAG_ID_IPTC_APP_EXPIRATION_TIME	Expiration time.
IGMDTAG_ID_IPTC_APP_SPECIAL_INSTRUCTIONS	Special instructions.
IGMDTAG_ID_IPTC_APP_ACTION_ADVISED	Action advised.
IGMDTAG_ID_IPTC_APP_REFERENCE_SERVICE	Reference service.
IGMDTAG_ID_IPTC_APP_REFERENCE_DATE	Reference date.
IGMDTAG_ID_IPTC_APP_REFERENCE_NUMBER	Reference number.
IGMDTAG_ID_IPTC_APP_DATE_CREATED	Date created.
IGMDTAG_ID_IPTC_APP_TIME_CREATED	Time created.
IGMDTAG_ID_IPTC_APP_DIGITAL_CREATION_DATE	Digital creation date.
IGMDTAG_ID_IPTC_APP_DIGITAL_CREATION_TIME	Digital creation time.
IGMDTAG_ID_IPTC_APP_ORIGINATING_PROGRAM	Originating program.
IGMDTAG_ID_IPTC_APP_PROGRAM_VERSION	Program version.
IGMDTAG_ID_IPTC_APP_OBJECT_CYCLE	Object cycle.
IGMDTAG_ID_IPTC_APP_BY_LINE	By-line information.
IGMDTAG_ID_IPTC_APP_BY_LINE_TITLE	By-line title.
IGMDTAG_ID_IPTC_APP_CITY	City information.
IGMDTAG_ID_IPTC_APP_SUBLOCATION	Sub-location information.
IGMDTAG_ID_IPTC_APP_PROVINCE_STATE	Province / State.
IGMDTAG_ID_IPTC_APP_COUNTRY_PRIMARY_LOC_CODE	Country/Primary location code.
IGMDTAG_ID_IPTC_APP_COUNTRY_PRIMARY_LOC_NAME	Country/Primary location name.
IGMDTAG_ID_IPTC_APP_ORIGINAL_TRANSM_REF	Original transmission reference.
IGMDTAG_ID_IPTC_APP_HEADLINE	Synopsis of the subject matter.
IGMDTAG_ID_IPTC_APP_CREDIT	Credit information.
IGMDTAG_ID_IPTC_APP_SOURCE	Source that identifies the original owner / creator.
IGMDTAG_ID_IPTC_APP_COPYRIGHT_NOTICE	Copyright notice.

IGMDTAG_ID_IPTC_APP_CONTACT	Contact information.
IGMDTAG_ID_IPTC_APP_CAPTION_ABSTRACT	Caption / Abstract.
IGMDTAG_ID_IPTC_APP_WRITER_EDITOR	Writer / Editor.
IGMDTAG_ID_IPTC_APP_RASTERIZED_CAPTION	Rasterized caption.
IGMDTAG_ID_IPTC_APP_IMAGE_TYPE	Image type.
IGMDTAG_ID_IPTC_APP_IMAGE_ORIENTATION	Image orientation.
IGMDTAG_ID_IPTC_APP_LANGUAGE_IDENTIFIER	Language identifier.
IGMDTAG_ID_IPTC_APP_AUDIO_TYPE	Audio type.
IGMDTAG_ID_IPTC_APP_AUDIO_SAMPLING_RATE	Audio sampling rate.
IGMDTAG_ID_IPTC_APP_AUDIO_SAMPLING_RESOLUTION	Audio sampling resolution.
IGMDTAG_ID_IPTC_APP_AUDIO_DURATION	Audio duration.
IGMDTAG_ID_IPTC_APP_AUDIO_OUTCUE	Audio outcue.
IGMDTAG_ID_IPTC_APP_OBJ_DATA_PREV_FILE_FORMAT	Object data preview file format.
IGMDTAG_ID_IPTC_APP_OBJ_DATA_PREV_FILE_FORMAT_VER	Object data preview file format version.
IGMDTAG_ID_IPTC_APP_OBJ_DATA_PREV_DATA	Object data preview data.

1.3.1.5.95 enumIGIPTCRecord3DatasetTags

Lists all IPTC Record 3 (Digital Newsphoto Parameter) DataSet tags.

Values:

IGMDTAG_ID_IPTC_PHOTO_RECORD_VERSION	Record version.
IGMDTAG_ID_IPTC_PHOTO_PICTURE_NUMBER	Picture number.
IGMDTAG_ID_IPTC_PHOTO_PIXELS_PER_LINE	Pixels per line.
IGMDTAG_ID_IPTC_PHOTO_NUMBER_OF_LINE	Number of line.
IGMDTAG_ID_IPTC_PHOTO_PIXEL_SIZE_SCAN_DIR	Pixel size in scanning direction.
IGMDTAG_ID_IPTC_PHOTO_PIXEL_SIZE_PERP_DIR	Pixel size perpendicular to scanning direction.
IGMDTAG_ID_IPTC_PHOTO_SUPPLEMENT_TYPE	Supplement type.
IGMDTAG_ID_IPTC_PHOTO_COLOUR_REPRESENTATION	Colour representation.
IGMDTAG_ID_IPTC_PHOTO_INTERCHANGE_COLOUR_SPACE	Interchange colour space.
IGMDTAG_ID_IPTC_PHOTO_COLOUR_SEQUENCE	Colour sequence.
IGMDTAG_ID_IPTC_PHOTO_ICC_INPUT_COLOUR_PROFILE	ICC input colour profile.
IGMDTAG_ID_IPTC_PHOTO_COLOUR_MATRIX_TABLE	Colour calibration matrix table.
IGMDTAG_ID_IPTC_PHOTO_LOOKUP_TABLE	Lookup table.
IGMDTAG_ID_IPTC_PHOTO_NUMBER_OF_INDEX_ENTRIES	Number of index entries.
IGMDTAG_ID_IPTC_PHOTO_COLOUR_PALETTE	Colour palette.
IGMDTAG_ID_IPTC_PHOTO_NUMBER_OF_BITS_PER_SAMPLE	Number of bits per sample.
IGMDTAG_ID_IPTC_PHOTO_SAMPLING_STRUCTURE	Sampling structure.
IGMDTAG_ID_IPTC_PHOTO_SCANNING_DIRECTION	Scanning direction.
IGMDTAG_ID_IPTC_PHOTO_IMAGE_ROTATION	Image rotation.
IGMDTAG_ID_IPTC_PHOTO_DATA_COMPRESSION_METHOD	Data compression method.
IGMDTAG_ID_IPTC_PHOTO_QUANTISATION_METHOD	Quantisation method.
IGMDTAG_ID_IPTC_PHOTO_END_POINTS	End points.
IGMDTAG_ID_IPTC_PHOTO_EXCURSION_TOLERANCE	Excursion tolerance.
IGMDTAG_ID_IPTC_PHOTO_BITS_PER_COMPONENT	Bits per component.
IGMDTAG_ID_IPTC_PHOTO_MAXIMUM_DENSITY_RANGE	Maximum density range.
IGMDTAG_ID_IPTC_PHOTO_GAMMA_COMPENSATED_VALUE	Gamma compensated value.

1.3.1.5.96 enumIGIPTCRecord7DatasetTags

Lists all IPTC Record 7 (Pre-Object) DataSet tags.

Values:

IGMDTAG_ID_IPTC_PREOBJ_SIZE_MODE	Size mode.
IGMDTAG_ID_IPTC_PREOBJ_MAX_SUBFILE_SIZE	Max subfile size.
IGMDTAG_ID_IPTC_PREOBJ_OBJ_DATA_SIZE_ANN	Object data size announced.
IGMDTAG_ID_IPTC_PREOBJ_MAX_OBJ_DATA_SIZE	Maximum object data size.

1.3.1.5.97 enumIGIPTCRecord8DatasetTags

Lists all IPTC Record 8 (Object) DataSet tags.

Values:

IGMDTAG_ID_IPTC_OBJ_SUBFILE	IPTC ObjectData subfile.

1.3.1.5.98 enumIGIPTCRecord9DatasetTags

Lists all IPTC Record 9 (Post-Object) DataSet tags.

Values:

IGMDTAG_ID_IPTC_POSTOBJ_CONFIRMED_OBJ_DATA_SIZE	Confirmed object data size.

1.3.1.5.99 enumIGIPTCRecordTags

Lists all IPTC Record tags.

Values:

IGMDTAG_ID_IPTC_ENVELOPE_RECORD	Envelope record.
IGMDTAG_ID_IPTC_APPLICATION_RECORD	Application record.
IGMDTAG_ID_IPTC_DIG_NEWS_PHOTO_PAR_RECORD	Digital Newsphoto Pararameter record.
IGMDTAG_ID_IPTC_PREOBJ_DESC_RECORD	Pre-object record.
IGMDTAG_ID_IPTC_OBJECT_RECORD	Object record.
IGMDTAG_ID_IPTC_POSTOBJ_DESC_RECORD	Post-object record.

1.3.1.5.100 enumIGIPTCTags

Lists all general IPTC tags.

Values:

IGMDTAG_ID_IPTC_FORMAT	IPTC metadata format identifier.

1.3.1.5.101 enumIGJPGTagIDs

Lists all JPEG tag identifiers.

Values:

IGMDTAG_ID_JPG_FORMAT	JPEG metadata format identifier.
IGMDTAG_ID_JPG_JFIF_HEADER	JFIF header.
IGMDTAG_ID_JPG_JFIF_VERSION	JFIF version.
IGMDTAG_ID_JPG_JFIF_UNITS	JFIF resolution unit.
IGMDTAG_ID_JPG_JFIF_X_RES	JFIF X resolution.
IGMDTAG_ID_JPG_JFIF_Y_RES	JFIF Y resolution.
IGMDTAG_ID_JPG_THUMB_WIDTH	JFIF thumbnail width.
IGMDTAG_ID_JPG_THUMB_HEIGHT	JFIF thumbnail height.
IGMDTAG_ID_JPG_THUMB_DATA	JFIF thumbnail data. Used internally.
IGMDTAG_ID_JPG_JFIF_EX_HEADER	JFIF extension header.
IGMDTAG_ID_JPG_JFIF_EX_CODE	JFIF extension code.
IGMDTAG_ID_JPG_FRAME_PRECISION	SOF Precision.
IGMDTAG_ID_JPG_FRAME_LINES	SOF Lines.
IGMDTAG_ID_JPG_FRAME_SAMPPL	SOF Samples Per Line.
IGMDTAG_ID_JPG_FRAME_COMPS	SOF Number of components.
IGMDTAG_ID_JPG_SCAN_COMPS	Number of components in the scan.
IGMDTAG_ID_JPG_SCAN_SP_START	Scan spectral start.
IGMDTAG_ID_JPG_SCAN_SP_END	Scan spectral end.
IGMDTAG_ID_JPG_SCAN_AH_AL	Scan AH, AL.
IGMDTAG_ID_JPG_PHOT_HEADER	Photoshop resources identifier.
IGMDTAG_ID_JPG_SOF0_SIZE	SOF0 segment size.
IGMDTAG_ID_JPG_SOF1_SIZE	SOF1 segment size.
IGMDTAG_ID_JPG_SOF2_SIZE	SOF2 segment size.
IGMDTAG_ID_JPG_SOF3_SIZE	SOF3 segment size.
IGMDTAG_ID_JPG_DHT_SIZE	DHT segment size.
IGMDTAG_ID_JPG_SOS_SIZE	SOS segment size.
IGMDTAG_ID_JPG_DQT_SIZE	DQT segment size.
IGMDTAG_ID_JPG_APP0_SIZE	APP0 segment size.
IGMDTAG_ID_JPG_APP1_SIZE	APP1 segment size.
IGMDTAG_ID_JPG_APP2_SIZE	APP2 segment size.
IGMDTAG_ID_JPG_APP3_SIZE	APP3 segment size.
IGMDTAG_ID_JPG_APP4_SIZE	APP4 segment size.
IGMDTAG_ID_JPG_APP5_SIZE	APP5 segment size.
IGMDTAG_ID_JPG_APP6_SIZE	APP6 segment size.
IGMDTAG_ID_JPG_APP7_SIZE	APP7 segment size.
IGMDTAG_ID_JPG_APP8_SIZE	APP8 segment size.
IGMDTAG_ID_JPG_APP9_SIZE	APP9 segment size.
IGMDTAG_ID_JPG_APP10_SIZE	APP10 segment size.
IGMDTAG_ID_JPG_APP11_SIZE	APP11 segment size.
IGMDTAG_ID_JPG_APP12_SIZE	APP12 segment size.
IGMDTAG_ID_JPG_APP13_SIZE	APP13 segment size.
IGMDTAG_ID_JPG_APP14_SIZE	APP14 segment size.

IGMDTAG_ID_JPG_APP15_SIZE	APP15 segment size.
IGMDTAG_ID_JPG_COM_SIZE	COM segment size.
IGMDTAG_ID_JPG_SOF0	Baseline DCT.
IGMDTAG_ID_JPG_SOF1	Extended sequential DCT.
IGMDTAG_ID_JPG_SOF2	Progressive DCT.
IGMDTAG_ID_JPG_SOF3	Lossless (sequential).
IGMDTAG_ID_JPG_DHT	Huffman tables.
IGMDTAG_ID_JPG_SOS	Start of Segment.
IGMDTAG_ID_JPG_DQT	Quantization tables.
IGMDTAG_ID_JPG_DRI	Restart Interval.
IGMDTAG_ID_JPG_APP0	Application marker - JFIF header (APP0).
IGMDTAG_ID_JPG_APP1	Application marker - first.
IGMDTAG_ID_JPG_APP2	Application marker - 2.
IGMDTAG_ID_JPG_APP3	Application marker - 3.
IGMDTAG_ID_JPG_APP4	Application marker - 4.
IGMDTAG_ID_JPG_APP5	Application marker - 5.
IGMDTAG_ID_JPG_APP6	Application marker - 6.
IGMDTAG_ID_JPG_APP7	Application marker - 7.
IGMDTAG_ID_JPG_APP8	Application marker - 8.
IGMDTAG_ID_JPG_APP9	Application marker - 9.
IGMDTAG_ID_JPG_APP10	Application marker - 10.
IGMDTAG_ID_JPG_APP11	Application marker - 11.
IGMDTAG_ID_JPG_APP12	Application marker - 12.
IGMDTAG_ID_JPG_APP13	Application marker - 13.
IGMDTAG_ID_JPG_APP14	Application marker - 14.
IGMDTAG_ID_JPG_APP15	Application marker - last.
IGMDTAG_ID_JPG_COM	Comment value.

1.3.1.5.102 enumIGJPGType

Identifies JPEG saving types.

Values:

IG_JPG_LOSSY	Lossy JPEG compression.
IG_JPG_LOSSLESS	Lossless JPEG compression.
IG_JPG_PROGRESSIVE	Progressive JPEG compression.

1.3.1.5.103 enumIGKFXTagIDs

Lists all KFX tag identifiers.

Values:

IGMDTAG_ID_KFX_FORMAT	KFX metadata format identifier.
IGMDTAG_ID_KFX_ID	Image identifier. R/O.
IGMDTAG_ID_KFX_HDR_SIZE	Header size. R/O.
IGMDTAG_ID_KFX_HDR_VER	Header version. R/O.
IGMDTAG_ID_KFX_IMAGE_ID	Image ID value. R/O.
IGMDTAG_ID_KFX_WIDTH	Image width. R/O.
IGMDTAG_ID_KFX_LENGTH	Image length. R/O.
IGMDTAG_ID_KFX_KFX_FORMAT	KFX format. R/O.
IGMDTAG_ID_KFX_BIT_SEX	Bit sex info. R/W.
IGMDTAG_ID_KFX_COLOR	Color info. R/W.
IGMDTAG_ID_KFX_XRES	Horizontal image resolution. R/W.
IGMDTAG_ID_KFX_YRES	Vertical image resolution. R/W.
IGMDTAG_ID_KFX_PLANES	Planes info. R/O.
IGMDTAG_ID_KFX_BITS_PER_PIX	Bits per pixel. R/O.
IGMDTAG_ID_KFX_PAPER_SIZE	Paper size. R/W.
IGMDTAG_ID_KFX_DATE_CRT	Creation date. R/W.
IGMDTAG_ID_KFX_DATE_MOD	Modification date. R/W.
IGMDTAG_ID_KFX_DATE_ACC	Access date. R/W.
IGMDTAG_ID_KFX_IDX_OFFSET	Index offset. R/W.
IGMDTAG_ID_KFX_IDX_LEN	Index length. R/W.
IGMDTAG_ID_KFX_COM_OFFSET	Com offset. R/W.
IGMDTAG_ID_KFX_COM_LEN	Com length. R/W.
IGMDTAG_ID_KFX_USER_OFFSET	User Offset. R/W.
IGMDTAG_ID_KFX_USER_LEN	User length. R/W.
IGMDTAG_ID_KFX_DATA_OFFSET	Data offset. R/W.
IGMDTAG_ID_KFX_DATA_LEN	Data length. R/W.

1.3.1.5.104 enumIGLicenseType

Identifies the different types of product license - evaluation, development, or deployment.

Values:

IG_VERSION_NONE	No license is available.
IG_VERSION_EVAL	Evaluation license.
IG_VERSION_DEVELOPMENT_ONLY	Development license.
IG_VERSION_DEPLOYMENT	Deployment license.

1.3.1.5.105 enumIGLVTagIDs

Lists all LV tag identifiers.

Values:

IGMDTAG_ID_LV_FORMAT	LV metadata format identifier.
IGMDTAG_ID_LV_YORIGIN	Image Y origin. R/W.
IGMDTAG_ID_LV_XORIGIN	Image X origin. R/W.
IGMDTAG_ID_LV_LINES	Number of lines. R/O.
IGMDTAG_ID_LV_PIXELS	Number of columns. R/O.
IGMDTAG_ID_LV_BITSPIX	Bits per pixel. R/O.
IGMDTAG_ID_LV_COMPRESSION	Image compression. R/O.
IGMDTAG_ID_LV_BYTEFORMAT	Byte format. R/O.
IGMDTAG_ID_LV_COMPVERSION	Comp version. R/O.
IGMDTAG_ID_LV_YAXIS	Y axis info. R/W.
IGMDTAG_ID_LV_XAXIS	X axis info. R/W.
IGMDTAG_ID_LV_NBLOCKTYPE	N Block type. R/W.
IGMDTAG_ID_LV_DISPLAYMETHOD	Display method. R/W.
IGMDTAG_ID_LV_XSEPERATION	X separation. R/W.
IGMDTAG_ID_LV_YSEPERATION	Y separation. R/W.
IGMDTAG_ID_LV_BLOCKLENGTH	Block length. R/W.
IGMDTAG_ID_LV_TEXT	Text info. R/W.

1.3.1.5.106 enumIGMergeModes

Identifies the type of arithmetic operation (merge method) that is performed on the values of all intersecting pixels resulting from the merge.

For example, if you set IG_ARITH_ADD merge method, the resulting pixel values (of those pixels that intersected from the two images) equal the sum of the value of the pixel in the original image and the value of pixel in the image being merged.

Values:

IG_ARITH_ADD	Img1 = Img1 + Img2
IG_ARITH_ADD_SIGN_CENTERED	Img1 = Img1+ SC_Img2
IG_ARITH_AND	Img1 = Img1 & Img2
IG_ARITH_DIVIDE	Img1 = Img1 / Img2
IG_ARITH_MULTI	Img1 = Img1 * Img2
IG_ARITH_NOT	Img1 = ~Img1
IG_ARITH_OR	Img1 = Img1 | Img2
IG_ARITH_OVER	Img1 = Img2
IG_ARITH_SUB	Img1 = Img1 - Img2
IG_ARITH_XOR	Img1 = Img1 ^ Img2

1.3.1.5.107 enumIGMETADItemType

Identifies the Metadata item type.

Values:

IG_METAD_LEVEL_START	Start of new metadata level.
IG_METAD_VALUE_ITEM	Metadata item.
IG_METAD_LEVEL_END	End of metadata level.

1.3.1.5.108 enumIGMSPTagIDs

Lists all MSP tag identifiers.

Values:

IGMDTAG_ID_MSP_FORMAT	MSP metadata format identifier.
IGMDTAG_ID_MSP_KEY1	Magic number.
IGMDTAG_ID_MSP_KEY2	Magic number.
IGMDTAG_ID_MSP_WIDTH	Width of the bitmap in pixels.
IGMDTAG_ID_MSP_HEIGHT	Height of the bitmap in pixels.
IGMDTAG_ID_MSP_X_AR_BITMAP	X Aspect ratio of the bitmap.
IGMDTAG_ID_MSP_Y_AR_BITMAP	Y Aspect ratio of the bitmap.
IGMDTAG_ID_MSP_X_AR_PRINTER	X Aspect ratio of the printer.
IGMDTAG_ID_MSP_Y_AR_PRINTER	Y Aspect ratio of the printer.
IGMDTAG_ID_MSP_X_PRINTER_WIDTH	Width of the printer in pixels.
IGMDTAG_ID_MSP_Y_PRINTER_HEIGHT	Height of the printer in pixels.
IGMDTAG_ID_MSP_X_ASPECT_CORR	X aspect correction (unused).
IGMDTAG_ID_MSP_Y_ASPECT_CORR	Y aspect correction (unused).
IGMDTAG_ID_MSP_CHECKSUM	Checksum of previous 24 bytes.
IGMDTAG_ID_MSP_PADDING	Unused padding.

1.3.1.5.109 enumIGMultInfo

Specifies attributes of multimedia images.

Values:

IG_MULT_INFO_HAS_VIDEO	TRUE value of the atribute indicates that the multimedia file has a video stream; otherwise it is FALSE.
IG_MULT_INFO_HAS_AUDIO	TRUE value of the atribute indicates that the multimedia file has an audio stream; otherwise it is FALSE.
IG_MULT_INFO_GIF_MIN_DELAY	Attribute value is a minimum delay of animated GIF frame display. This value is applied to the frame if its own delay is less then value of IG_MULT_INFO_GIF_MIN_DELAY_THRESHOLD attribute.
IG_MULT_INFO_GIF_MIN_DELAY_THRESHOLD	Attribute value is a threshold for minimum delay of animated GIF frame display. Frames with delay less than this value are displayed with delay specified by IG_MULT_INFO_GIF_MIN_DELAY attribute.

Remarks:

Info IDs for overall and per-frame info common to all multimedia sources IDs are allocated as follows: 0 - 1999 = Overall info common to all multimedia sources 2000 - 3999 = Per-frame info common to all multimedia sources 4000 - 5999 = Overall info specific to individual sources (overlap is fine) 6000 - 7999 = Per-frame info specific to individual sources (overlap is fine) 8000 - ???? = reserved

1.3.1.5.110 enumIGNCRTagIDs

Lists all NCR tag identifiers.

Values:

IGMDTAG_ID_NCR_FORMAT	NCR metadata format identifier.
IGMDTAG_ID_NCR_DATA_FORMAT	Data format.
IGMDTAG_ID_NCR_OPTIONS	Options info.
IGMDTAG_ID_NCR_ENCRYPTION	Encryption info.
IGMDTAG_ID_NCR_AUTHENTICATION	Authentication info.
IGMDTAG_ID_NCR_AUTH_MAC	Authentication Mac info.
IGMDTAG_ID_NCR_DATA_SIZE	Data size.
IGMDTAG_ID_NCR_REAL_BPP	Real bpp value.
IGMDTAG_ID_NCR_STORE_BPP	Store bpp.
IGMDTAG_ID_NCR_REAL_WIDTH	Real width.
IGMDTAG_ID_NCR_STORE_WIDTH	Store width.
IGMDTAG_ID_NCR_REAL_HEIGHT	Real height.
IGMDTAG_ID_NCR_STORE_HEIGHT	Store height.
IGMDTAG_ID_NCR_ORIENT	Orientation setting.
IGMDTAG_ID_NCR_PMI	Invert pixels flag.
IGMDTAG_ID_NCR_DATA_ENDIAN	Data endian.
IGMDTAG_ID_NCR_FILL_ORDER	Fill order.
IGMDTAG_ID_NCR_GRANULARITY	Granularity info.
IGMDTAG_ID_NCR_MIN_PIX_VALUE	Min pix value.
IGMDTAG_ID_NCR_MAX_PIX_VALUE	Max pix value.
IGMDTAG_ID_NCR_X_RES	Horizontal resolution.
IGMDTAG_ID_NCR_Y_RES	Vertical resolution.
IGMDTAG_ID_NCR_RES_UNIT	Resolution unit.
IGMDTAG_ID_NCR_ERROR	Error info.

1.3.1.5.111 enumIGNoiseMethods

This enumeration contains noise methods for IG_FX_noise.

Values:

IG_NOISE_LINEAR	Linear method.
IG_NOISE_GAUSSIAN	Gaussian method.

1.3.1.5.112 enumIGOrientationModes

Identifies how the image is oriented before it is drawn on the output device. Possible values are determined by the constants of the form IG_DSPL_ORIENT_X_Y, where each of X and Y can be LEFT, TOP, RIGHT or BOTTOM. X represents the position of the topmost row of the bitmap after applying the transformation. Y represents the position of the left-most column of the bitmap after applying the transformation.

For example, IG_DSPL_ORIENT_RIGHT_TOP means that the left-most column becomes the image's new topmost row, and that the topmost row becomes the image's new right-most column. The image, therefore, is rotated on 90 degrees.

Values:

IG_DSPL_ORIENT_BOTTOM_LEFT	The image is flipped vertically.
IG_DSPL_ORIENT_BOTTOM_RIGHT	The image is rotated 180 degrees.
IG_DSPL_ORIENT_LEFT_BOTTOM	The image is rotated 270 degrees.
IG_DSPL_ORIENT_LEFT_TOP	The image is rotated 270 degrees and then flipped vertically.
IG_DSPL_ORIENT_RIGHT_BOTTOM	The image is rotated 90 degrees and then flipped vertically.
IG_DSPL_ORIENT_RIGHT_TOP	The image is rotated 90 degrees.
IG_DSPL_ORIENT_TOP_LEFT	The image is displayed unchanged.
IG_DSPL_ORIENT_TOP_RIGHT	The image is flipped horizontally.

1.3.1.5.113 enumIGPaletteFormats

Identifies the different formats used for storing image palette in the external file.

Values:

IG_PALETTE_FORMAT_INVALID	Returned when a file could not be read.
IG_PALETTE_FORMAT_RAW_BGR	This is the raw DIB format BGR.
IG_PALETTE_FORMAT_RAW_BGRQ	This is the raw DIB format BGRQ.
IG_PALETTE_FORMAT_RAW_RGB	This is the raw DIB format RGB.
IG_PALETTE_FORMAT_RAW_RGBQ	This is the raw DIB format RGBQ.
IG_PALETTE_FORMAT_TEXT	ASCII text file.
IG_PALETTE_FORMAT_HALO_CUT	Dr Halo .PAL file for use with a CUT file format.

1.3.1.5.114 enumIGPBMTagIDs

Lists all PBM tag identifiers.

Values:

IGMDTAG_ID_PBM_FORMAT	PBM metadata format identifier.

1.3.1.5.115 enumIGPCDTagIDs

Lists all PCD tag identifiers.

Values:

IGMDTAG_ID_PCD_FORMAT	PCD metadata format identifier.
IGMDTAG_ID_PCD_IPICA_RESERVED	IPICA reserved.
IGMDTAG_ID_PCD_IPICA_IMAGE_PACK_PARAMS	IPICA image pack params.
IGMDTAG_ID_PCD_IPICA_BASE4_STOP_OFFSET	IPICA base4 stop offset.
IGMDTAG_ID_PCD_IPICA_BASE16_STOP_OFFSET	IPICA base16 stop offset.
IGMDTAG_ID_PCD_IPICA_IPE_STOP_OFFSET	IPICA IPE stop offset.
IGMDTAG_ID_PCD_IPICA_IP_INTERLEAVE_RATIO	IPICA IP interleave ratio.
IGMDTAG_ID_PCD_IPI_SIGNATURE	IPI signature.
IGMDTAG_ID_PCD_IPI_VERSION_NUMBER	IPI version number.
IGMDTAG_ID_PCD_IPI_SOFTWARE_RELEASE	IPI software release.
IGMDTAG_ID_PCD_IPI_IMAGE_MAG_DESCRIPTION	IPI image mag description.
IGMDTAG_ID_PCD_IPI_IMAGE_SCAN_TIME	IPI image scan time.
IGMDTAG_ID_PCD_IPI_LAST_MODIFICATION_DATE	IPI last modification date.
IGMDTAG_ID_PCD_IPI_MED_ORIGINAL_RECORDING	IPI med original recording.
IGMDTAG_ID_PCD_IPI_TYPE_ORIGINAL_RECORDING	IPI type original recording.
IGMDTAG_ID_PCD_IPI_SCANNER_VENDOR	IPI scanner vendor.
IGMDTAG_ID_PCD_IPI_SCANNER_PRODUCT	IPI scanner product.
IGMDTAG_ID_PCD_IPI_SCANNER_FIRMWARE_LEVEL	IPI scanner firmware level.
IGMDTAG_ID_PCD_IPI_SCANNER_FIRMWARE_DATE	IPI scanner firmware date.
IGMDTAG_ID_PCD_IPI_SCANNER_SERIAL_NUMBER	IPI scanner serial number.
IGMDTAG_ID_PCD_IPI_SCANNER_PIXEL_SIZE	IPI scanner pixel size.
IGMDTAG_ID_PCD_IPI_EQUIPMENT_MANUFACTURER	IPI equipment manufacturer.
IGMDTAG_ID_PCD_IPI_PHOTONAME_CHAR_SET	IPI photoname char set.
IGMDTAG_ID_PCD_IPI_PHOTONAME_ESC_SEQ	IPI photoname esc seq.
IGMDTAG_ID_PCD_IPI_PHOTONAME	IPI photoname.
IGMDTAG_ID_PCD_IPI_SBA_DATA	IPI SBA data.
IGMDTAG_ID_PCD_IPI_COPYRIGHT_STATUS	IPI copyright status.
IGMDTAG_ID_PCD_IPI_COPYRIGHT_FILENAME	IPI copyright filename.

1.3.1.5.116 enumIGPCXTagIDs

Lists all PCX tag identifiers.

Values:

IGMDTAG_ID_PCX_FORMAT	PCX metadata format identifier.
IGMDTAG_ID_PCX_MANUFACTURER	Manufacturer magic value.
IGMDTAG_ID_PCX_VERSION_INFO	Version info.
IGMDTAG_ID_PCX_ENCODE	Encoding type.
IGMDTAG_ID_PCX_BIT_PER_PLANE	Bit per plane.
IGMDTAG_ID_PCX_X1	Left image coordinate to display.
IGMDTAG_ID_PCX_Y1	Top image coordinate to display.
IGMDTAG_ID_PCX_X2	RIght image coordinate to display.
IGMDTAG_ID_PCX_Y2	Bottom image coordinate to display.
IGMDTAG_ID_PCX_H_RES	Horizontal Resolution of creating device.
IGMDTAG_ID_PCX_V_RES	Vertical Resolution of creating device.
IGMDTAG_ID_PCX_PALETTE_TABLE	Palette table.
IGMDTAG_ID_PCX_VIDEO_MODE	Video mode.
IGMDTAG_ID_PCX_NUM_OF_PLANES	Number of planes.
IGMDTAG_ID_PCX_BYTES_PER_LINE	Bytes per line.
IGMDTAG_ID_PCX_PALETTE_INFO	Palette info.
IGMDTAG_ID_PCX_SCANNER_H_RES	Scanner H_Res.
IGMDTAG_ID_PCX_SCANNER_V_RES	Scanner V_Res.
IGMDTAG_ID_PCX_EXTRA	Extra data length.

1.3.1.5.117 enumIGPixAccessMode

Specifies pixel data formats used by pixel access functions.

Values:

IG_PIX_ACCESS_MODE_LEGACY	Legacy pixel data format, native for ImageGear versions prior 14.5.
IG_PIX_ACCESS_MODE_NEW	New pixel data format, available in ImageGear version 14.5 and beyond.

1.3.1.5.118 enumIGPNGTagIDs

Lists all PNG tag identifiers.

Values:

IGMDTAG_ID_PNG_FORMAT	PNG metadata format identifier.
IGMDTAG_ID_PNG_HEADER	Header information.
IGMDTAG_ID_PNG_TRANSPARENCY	Transparency information.
IGMDTAG_ID_PNG_GAMMA	Gamma information.
IGMDTAG_ID_PNG_CHROMATICITIES	Chromaticities information.
IGMDTAG_ID_PNG_SRGB	Standard RGB information.
IGMDTAG_ID_PNG_ICC_PROFILE	ICC profile information.
IGMDTAG_ID_PNG_BACKGROUND	Background information.
IGMDTAG_ID_PNG_SIGNIFICANT_BITS	Significant bits information.
IGMDTAG_ID_PNG_SUGGESTED_PALETTE	Suggested palette information.
IGMDTAG_ID_PNG_HISTOGRAM	Histogram information.
IGMDTAG_ID_PNG_TIME	Time image last modification information.
IGMDTAG_ID_PNG_TEXT	Text data information.
IGMDTAG_ID_PNG_COMPRESSED_TEXT	Compressed textual data information.
IGMDTAG_ID_PNG_INTERNATIONAL_TEXT	International textual data information.
IGMDTAG_ID_PNG_CALIBRATION	Calibration information.
IGMDTAG_ID_PNG_PHYSICAL_SCALE	Physical scale information.
IGMDTAG_ID_PNG_GIF_APP_EXT	GIF application extension information.
IGMDTAG_ID_PNG_GIF_CONTROL	GIF control information.
IGMDTAG_ID_PNG_IMAGE_OFFSET	Image offset.
IGMDTAG_ID_PNG_FRACTAL_PARAMETERS	Fractal parameters information.
IGMDTAG_ID_PNG_GIF_TEXT_EXT	Gif text extension information.
IGMDTAG_ID_PNG_RESOLUTION	Resolution information.

1.3.1.5.119 enumIGPromotionModes

This enumeration specifies color promotion modes.

Values:

IG_PROMOTE_TO_4	Promote to 4-bit Indexed.
IG_PROMOTE_TO_8	Promote to 8-bit Indexed.
IG_PROMOTE_TO_24	Promote to 24-bit RGB.
IG_PROMOTE_TO_32	Promote to 32-bit CMYK.

1.3.1.5.120 enumIGPSDTagIDs

Lists all PSD tag identifiers. See "Photoshop CS File Formats Specification" available in Adobe Photoshop SDK for more details.

Values:

IGMDTAG_ID_PSD_FORMAT	PSD metadata format identifier.
IGMDTAG_ID_PSD_SIGNATURE	File type identifier.
IGMDTAG_ID_PSD_VERSION	Version number.
IGMDTAG_ID_PSD_ROWS	The height of the image in pixels.
IGMDTAG_ID_PSD_COLS	The width of the image in pixels.
IGMDTAG_ID_PSD_DEPTH	The number of bits per channel.
IGMDTAG_ID_PSD_MODE	Color mode.
IGMDTAG_ID_PSD_MODE_LEN	Color mode data length.
IGMDTAG_ID_PSD_COMPRESSION	Image data compression.
IGMDTAG_ID_PSD_FILE_HDR	File header.
IGMDTAG_ID_PSD_COLOR_DATA	Color data.
IGMDTAG_ID_PSD_RESOURCE	Photoshop resources.
IGMDTAG_ID_PSD_LAYER	Layers data.
IGMDTAG_ID_PSD_GLOBAL_MASK	Extra layers data.
IGMDTAG_ID_PSD_LAYER_RECT	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_LAYER_NUMBER_CHANNELS	Number of channels in the layer.
IGMDTAG_ID_PSD_LAYER_BLEND_MODE_KEY	Blend mode key.
IGMDTAG_ID_PSD_LAYER_OPACITY	Layer opacity (0 = transparent ... 255 = opaque).
IGMDTAG_ID_PSD_LAYER_CLIPPING	Layer clipping (0 = base, 1 = non-base).
IGMDTAG_ID_PSD_LAYER_FLAGS	Flags: bit 0 = transparency protected; bit 1 = visible; bit 2 = obsolete; bit 3 = 1 for Photoshop 5.0 and later, tells if bit 4 has useful information; bit 4 = pixel data irrelevant to appearance of document.
IGMDTAG_ID_PSD_LAYER_MASK_DATA	Mask data.
IGMDTAG_ID_PSD_LAYER_ASCII_NAME	Layer name as ASCII string.
IGMDTAG_ID_PSD_LAYER_CHANNEL_LEN_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_LAYER_BLENDING_RANGES_DATA	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_NUM_LAYERS	Number of layers.
IGMDTAG_ID_PSD_GLOBAL_MASK_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_ADJUSTMENT_LAYER_INFO	Adjustment info.
IGMDTAG_ID_PSD_LAYER_UNICODE_NAME	Layer name as Unicode string.
IGMDTAG_ID_PSD_LAYER_ID	Layer ID.
IGMDTAG_ID_PSD_EFFECT_LAYER_INFO	Effect Layer info.
IGMDTAG_ID_PSD_EFFECT_LAYER_COMMON_STATE_INFO	Effects layer, common state info.
IGMDTAG_ID_PSD_EFFECT_LAYER_SHADOW_INFO	Effects layer, drop shadow and inner shadow info.
IGMDTAG_ID_PSD_EFFECT_LAYER_GLOW_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_EFFECT_LAYER_BEVEL_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_TOOL_TYPE_INFO	Type Tool Info.
IGMDTAG_ID_PSD_PATTERN	Pattern fill setting.

IGMDTAG_ID_PSD_ANNOTATIONS	Annotations.
IGMDTAG_ID_PSD_BLEND_CLIPPING_ELEMENTS	Blend clipping elements.
IGMDTAG_ID_PSD_BLEND_INTERIOR_ELEMENTS	Blend interior elements.
IGMDTAG_ID_PSD_KNOCKOUT_SETTING	Knockout setting.
IGMDTAG_ID_PSD_PROTECTED_SETTING	Protected setting.
IGMDTAG_ID_PSD_SHEET_COLOR_SETTING	Sheet color setting.
IGMDTAG_ID_PSD_REFERENCE_POINT	Reference point.
IGMDTAG_ID_PSD_OBJ_BASED_EFFECTS_LAYER_INFO	Object-based effects layer info.
IGMDTAG_ID_PSD_GRADIENT_SETTINGS	Gradient settings.
IGMDTAG_ID_PSD_EXTRA_LAYERS_DATA	Extra layers data.
IGMDTAG_ID_PSD_PhotoshopAdditionalLayerInfo	Additional layer info.
IGMDTAG_ID_PSD_PhotoshopAdditionalLayerInfoKey	Additional layer key.
IGMDTAG_ID_PSD_PhotoshopAdditionalLayerInfoData	Additional layer data.
IGMDTAG_ID_PSD_PhotoshopAdditionalLayerInfoTag	Additional layer tag.
IGMDTAG_ID_PSD_PhotoshopAdditionalLayerInfoDescr	Additional layer description.
IGMDTAG_ID_PSD_HEADER	Header data section.
IGMDTAG_ID_PSD_PHOTOSHOP_RESOURCES	Photoshop resources.
IGMDTAG_ID_PSD_PHOTOSHOP_IMG_RESOURCE	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_PHOTOSHOP_IMG_RESOURCE_DATA	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_PHOTOSHOP_IMG_RESOURCE_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_PHOTOSHOP_IMG_RESOURCE_ID	This value has been deprecated and will be removed from the public API in a future release.
IGMDTAG_ID_PSD_LAYER_INFO	Layer info.
IGMDTAG_ID_PSD_RECTANGLE	Layer rectangle.
IGMDTAG_ID_PSD_RECT_LEFT	Layer rectangle left.
IGMDTAG_ID_PSD_RECT_TOP	Layer rectangle top.
IGMDTAG_ID_PSD_RECT_RIGTH	Layer rectangle right.
IGMDTAG_ID_PSD_RECT_BOTTOM	Layer rectangle bottom.
IGMDTAG_ID_PSD_LAYER_CHANNELS_IDS	Layer rectangle IDs.
IGMDTAG_ID_PSD_LAYER_EXTRA_DATA	Layer extra data.
IGMDTAG_ID_PSD_LAYER_EXTRA_DATA_REC	Layer extra data items.
IGMDTAG_ID_PSD_LAYERS	PSD layers data.

1.3.1.5.121 enumIGRASTagIDs

Lists all RAS tag identifiers.

Values:

IGMDTAG_ID_RAS_FORMAT	RAS metadata format identifier.
IGMDTAG_ID_RAS_MAGIC	RAS magic value.
IGMDTAG_ID_RAS_WIDTH	Image width.
IGMDTAG_ID_RAS_HEIGHT	Image height.
IGMDTAG_ID_RAS_DEPTH	Image depth.
IGMDTAG_ID_RAS_LENGTH	Length of the image data (which is the length of the file minus the length of the header and colormap).
IGMDTAG_ID_RAS_TYPE	RAS format type.
IGMDTAG_ID_RAS_COLOR_MAP_TYPE	Color map type.
IGMDTAG_ID_RAS_COLOR_MAP_LENGTH	Color map length.

1.3.1.5.122 enumIGResampleInModes

This enumeration contains input modes for IG_FX_pixelate method.

Values:

IG_RESAMPLE_IN_AVE	Average.
IG_RESAMPLE_IN_MIN	Min.
IG_RESAMPLE_IN_MAX	Max.
IG_RESAMPLE_IN_CENTER	Center.

1.3.1.5.123 enumIGResampleOutModes

This enumeration contains output modes for IG_FX_pixelate method.

Values:

IG_RESAMPLE_OUT_SQUARE	Square.
IG_RESAMPLE_OUT_CIRCLE	Circle.

1.3.1.5.124 enumIGResolutionUnits

Identifies the different resolution units.

Values:

IG_RESOLUTION_NO_ABS	No absolute units.
IG_RESOLUTION_METERS	Pels (Pixels) Per Meter.
IG_RESOLUTION_INCHES	Dots (Pixels) Per Inch.
IG_RESOLUTION_CENTIMETERS	Pixels Per Centimeter.
IG_RESOLUTION_10_INCHES	Dots (Pixels) Per 10 Inches.
IG_RESOLUTION_10_CENTIMETERS	Pixels Per 10 Centimeters.
IG_RESOLUTION_LAST	

1.3.1.5.125 enumIGRotationModes

This enumeration specifies modes of image rotation.

Values:

IG_ROTATE_CLIP	Clip the rotated image to keep the image bitmap the same size.
IG_ROTATE_EXPAND	Expand the size of the bitmap if necessary to retain the entire rotated image.

1.3.1.5.126 enumIGRotationValues

This enumeration specifies angles of image rotation by a multiple of 90 degrees.

Values:

IG_ROTATE_0	No rotation.
IG_ROTATE_90	Rotation by 90 degrees.
IG_ROTATE_180	Rotation by 180 degrees.
IG_ROTATE_270	Rotation by 270 degrees.

1.3.1.5.127 enumIGSaveFormats

Identifies the formats available for saving.

Values:

IG_SAVE_BMP_RLE	Microsoft Windows bitmap with RLE compression
IG_SAVE_BMP_UNCOMP	Microsoft Windows bitmap uncompressed
IG_SAVE_BRK_G3	BTR with Group 3 compression
IG_SAVE_BRK_G3_2D	BTR with Group 3 2D compression
IG_SAVE_CAL	CAL
IG_SAVE_CGM	CGM
IG_SAVE_CLP	CLP
IG_SAVE_DCM	DICOM
IG_SAVE_DCX	Paintbrush
IG_SAVE_DWF	DWF
IG_SAVE_DWG	DWG
IG_SAVE_DXF	DXF
IG_SAVE_EPS_G3	Encapsulated postscript with Group 3 compression
IG_SAVE_EPS_G4	Encapsulated postscript with Group 4 compression
IG_SAVE_EPS_JPG	Encapsulated postscript with JPG compression
IG_SAVE_EPS_UNCOMP	Encapsulated postscript uncompressed
IG_SAVE_EXIF_JPEG	Exchangeable image file format
IG_SAVE_EXIF_TIFF	Exchangeable image file format (EXIF-TIFF)
IG_SAVE_FPX_JPG	FlashPix with JPEG compression
IG_SAVE_FPX_NOCHANGE	FlashPix with the current compression
IG_SAVE_FPX_SINCOLOR	FlashPix with the single color compression
IG_SAVE_FPX_UNCOMP	FlashPix uncompressed
IG_SAVE_GIF	GIF
IG_SAVE_ICA_G3	IBM IOCA with Group 3 compression
IG_SAVE_ICA_G4	IBM IOCA with Group 4 compression
IG_SAVE_ICA_IBM_MMR	IBM IOCA with IBM MMR compression
IG_SAVE_ICO	windows icon
IG_SAVE_IFF	Interchange uncompressed
IG_SAVE_IFF_RLE	Interchange with RLE compression
IG_SAVE_IMT	IMT
IG_SAVE_JB2	Reserved for future use.
IG_SAVE_JBIG	JBIG
IG_SAVE_JPEG2K	JPEG2000
IG_SAVE_JPG	JPEG File Interchange
IG_SAVE_JPX	JPX
IG_SAVE_LURADOC	This value has been deprecated and will be removed from the public API in a future release.
IG_SAVE_LURAJP2	This value has been deprecated and will be removed from the public API in a future release.
IG_SAVE_LURAWAVE	This value has been deprecated and will be removed from the public API in a future release.
IG_SAVE_MOD_G3	IBM MO:DCA with Group 3 compression
IG_SAVE_MOD_G4	IBM MO:DCA with Group 4 compression

IG_SAVE_MOD_IBM_MMR	IBM MO:DCA with MMR compression
IG_SAVE_NCR	NCR uncompressed
IG_SAVE_NCR_G4	NCR with Group 4 compression
IG_SAVE_PBM_ASCII	PBM converted to ASCII text
IG_SAVE_PBM_RAW	PBM binary row format
IG_SAVE_PCT	Mac Pict
IG_SAVE_PCX	PC Paintbrush File Format
IG_SAVE_PDF_DEFLATE	Adobe PDF with Deflate compression
IG_SAVE_PDF_G3	Adobe PDF with Group 3 compression
IG_SAVE_PDF_G3_2D	Adobe PDF with Group3 2D compression
IG_SAVE_PDF_G4	Adobe PDF with Group 4 compression
IG_SAVE_PDF_JPG	Adobe PDF with JPEG compression
IG_SAVE_PDF_LZW	Adobe PDF with LZW compression
IG_SAVE_PDF_RLE	Adobe PDF with RLE compression
IG_SAVE_PDF_UNCOMP	Adobe PDF uncompressed
IG_SAVE_PJPEG	Not supported
IG_SAVE_PNG	Portable network graphics
IG_SAVE_PS_DEFLATE	Postscript with Deflate compression
IG_SAVE_PS_G3	Postscript with Group 3 compression
IG_SAVE_PS_G3_2D	Postscript with Group3 2D compression
IG_SAVE_PS_G4	Postscript with Group 4 compression
IG_SAVE_PS_JPG	Postscript with JPEG compression
IG_SAVE_PS_LZW	Postscript with LZW compression
IG_SAVE_PS_RLE	Postscript with RLE compression
IG_SAVE_PS_UNCOMP	Postscript uncompressed
IG_SAVE_PSB	Adobe PSB
IG_SAVE_PSB_PACKED	Adobe PSB with packed bits compression
IG_SAVE_PSD	Adobe PSD
IG_SAVE_PSD_PACKED	Adobe PSD with packed bits compression
IG_SAVE_RAS	RAS
IG_SAVE_RAW_G3	RAW with Group 3 compression
IG_SAVE_RAW_G32D	RAW with Group 3 2D compression
IG_SAVE_RAW_G4	RAW with Group 4 compression
IG_SAVE_RAW_LZW	RAW with LZW compression
IG_SAVE_RAW_RLE	RAW with RLE compression
IG_SAVE_SCI_ST	Scitex CT format
IG_SAVE_SGI	SGI
IG_SAVE_SGI_RLE	SGI with RLE compression
IG_SAVE_SVG	SVG
IG_SAVE_TGA	TGA
IG_SAVE_TGA_RLE	TGA with RLE compression
IG_SAVE_TIF_G3	Tagged Image File Format with Group 3 compression
IG_SAVE_TIF_G3_2D	Tagged Image File Format with Group 3 2D compression
IG_SAVE_TIF_G4	Tagged Image File Format with Group 4 compression
IG_SAVE_TIF_HUFFMAN	Tagged Image File Format with Huffman compression
IG_SAVE_TIF_JPG	Tagged Image File Format with JPEG compression
IG_SAVE_TIF_LZW	Tagged Image File Format with LZW compression
IG_SAVE_TIF_PACKED	Tagged Image File Format with Packed Bits compression

IG_SAVE_TIF_UNCOMP	Tagged Image File Format uncompressed
IG_SAVE_U3D	U3D
IG_SAVE_UNKNOWN	Unknown format
IG_SAVE_WBMP	Wireless Bitmap File Format
IG_SAVE_WL16	Not supported
IG_SAVE_WLT	Not supported
IG_SAVE_WMF	Windows MetaFile
IG_SAVE_XBM	XBM
IG_SAVE_XPM	XPM
IG_SAVE_XPS	XPS (XML Paper Specification)
IG_SAVE_XWD	XWD

1.3.1.5.128 enumIGSCICTTagIDs

Lists all SCI_CT tag identifiers.

Values:

IGMDTAG_ID_SCICT_FORMAT	SCI_CT metadata format identifier.

1.3.1.5.129 enumIGSGITagIDs

Lists all SGI tag identifiers.

Values:

IGMDTAG_ID_SGI_FORMAT	SGI metadata format identifier.
IGMDTAG_ID_SGI_MAGIC	IRIS image file magic number. R/O.
IGMDTAG_ID_SGI_STORAGE	Storage format. R/O.
IGMDTAG_ID_SGI_BPC	Number of bytes per pixel channel. R/O.
IGMDTAG_ID_SGI_DIMENSION	Number of dimensions. R/O.
IGMDTAG_ID_SGI_X_SIZE	X size in pixels. R/O.
IGMDTAG_ID_SGI_Y_SIZE	Y size in pixels. R/O.
IGMDTAG_ID_SGI_Z_SIZE	Number of channels. R/O.
IGMDTAG_ID_SGI_PIX_MIN	Minimum pixel value. R/W.
IGMDTAG_ID_SGI_PIX_MAX	Maximum pixel value. R/W.
IGMDTAG_ID_SGI_DUMMY1	Dummy 1 value. R/W.
IGMDTAG_ID_SGI_IMAGE_NAME	Image name. R/W.
IGMDTAG_ID_SGI_COLOR_MAP	Color map. R/O.
IGMDTAG_ID_SGI_DUMMY2	Dummy 2 value. R/W.

1.3.1.5.130 enumIGSysDataType

Identifies the ImageGear data types.

Values:

AM_TID_META_INT8	Data type is 8 bit signed integer.
AM_TID_META_UINT8	Data type is 8 bit unsigned integer.
AM_TID_META_INT16	Data type is 16 bit signed integer.
AM_TID_META_UINT16	Data type is 16 bit unsigned integer.
AM_TID_META_INT32	Data type is 32 bit signed integer.
AM_TID_META_UINT32	Data type is 32 bit unsigned integer.
AM_TID_META_BOOL	Data type is Boolean.
AM_TID_META_STRING	Data type is String.
AM_TID_META_RATIONAL_UINT32	Data type is Rational 32 bit unsigned integer.
AM_TID_META_RATIONAL_INT32	Data type is Rational 32 bit bit signed integer.
AM_TID_META_FLOAT	Data type is Float.
AM_TID_META_DOUBLE	Data type is Double.
AM_TID_RAW_DATA	Data type is Raw Data.
AM_TID_META_INT64	Data type is 64 bit signed integer.
AM_TID_META_UINT64	Data type is 64 bit unsigned integer.
AM_TID_META_STRING32	Data type is a String holding up to 32 characters. Corresponds to TW_STR32 data type of TWAIN API specification.
AM_TID_META_STRING64	Data type is a String holding up to 64 characters. Corresponds to TW_STR64 data type of TWAIN API specification.
AM_TID_META_STRING128	Data type is a String holding up to 128 characters. Corresponds to TW_STR128 data type of TWAIN API specification.
AM_TID_META_STRING255	Data type is a String holding up to 254 characters. Corresponds to TW_STR255 data type of TWAIN API specification.
AM_TID_META_STRING1024	Data type is a String holding up to 1024 characters. Corresponds to TW_STR1024 data type of TWAIN API specification.
AM_TID_META_STRING_UNICODE512	Data type is a String holding up to 512 unicode (wchar_t) characters. Corresponds to TW_UNI512 data type of TWAIN API specification.
AM_TID_META_DRECT	Data type is a AT_DRECT.

1.3.1.5.131 enumIGTagConstants

This enumeration has been deprecated and will be removed from the public API in a future release. Please use enumIGBMPTagIDs instead.

Values:

IGTAGVAL_BMP_TYPE_BMC	This value has been deprecated and will be removed from the public API in a future release. Please use IG_BMP_TYPE_BMC instead.
IGTAGVAL_BMP_TYPE_BMI	This value has been deprecated and will be removed from the public API in a future release. Please use IG_BMP_TYPE_BMI instead.
IGTAGVAL_BMP_TYPE_BMI2	This value has been deprecated and will be removed from the public API in a future release. Please use IG_BMP_TYPE_BMI2 instead.
IGTAGVAL_PCT_VERSION_1	This value has been deprecated and will be removed from the public API in a future release. Please use IG_PCT_VERSION_1 instead.
IGTAGVAL_PCT_VERSION_2	This value has been deprecated and will be removed from the public API in a future release. Please use IG_PCT_VERSION_2 instead.

1.3.1.5.132 enumIGTags

This enumeration has been deprecated and will be removed from the public API in a future release. Please use ImageGear metadata callbacks API instead.

Values:

IGTAG_BMP_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_PLANES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_BITCOUNT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_COMPRESSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_XPELSPERMETER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_YPELSPERMETER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_CLRUSED	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_CLRIMPORTANT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_UNITS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_RECORDING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_RENDERING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_SIZE1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_SIZE2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_COLORENCODING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_IDENTIFIER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_TYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_REDMASK	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_GREENMASK	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_BLUEMASK	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_ALPHAMASK	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_CSTYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_ENDPNTCOORDS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_GAMMARED	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_BMP_GAMMAGREEN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BMP_GAMMABLUE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_JFIF_ID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_JFIF_VERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_JFIF_UNITS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_JFIF_X_RESOLUTION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_JFIF_Y_RESOLUTION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_JFIF_THUMBNAIL_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_JFIF_THUMBNAIL_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_COMMENT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_QUANT1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_QUANT2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_QUANT3	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_QUANT4	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_PRECISION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_LINES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_SAMPLES_PER_LINE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_COMPONENTS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_COMPID1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_COMPID2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_COMPID3	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_MCU_HV1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_MCU_HV2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_MCU_HV3	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_QUANT1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_QUANT2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_QUANT3	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_SCAN_COMPONENTS	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_JPG_SCAN_COMP_SELECT1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_SCAN_COMP_SELECT2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_SCAN_COMP_SELECT3	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_SCAN_DC_AC1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_SCAN_DC_AC2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_SCAN_DC_AC3	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_SCAN_SPECT_START	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_SCAN_SPECT_END	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_SCAN_AH_AL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_APPDATA	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_APPDATA_LAST	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_JPG_FRAME_MARKER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_SPECVERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_SRCDOCID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_DSTDOCID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_TXTFILID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_FIGID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_RTYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_RORIENT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_RPELCNT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_RDENSITY	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_SRCGPH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_DOCCLS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_FOSIPUBID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CAL_NOTES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_MANUFACTURER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_VERSION_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_ENCODE	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_PCX_BIT_PER_PLANE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_X1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_Y1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_X2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_Y2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_H_RES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_V_RES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_PALETTE_TABLE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_VIDEO_MODE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_NUM_OF_PLANES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_BYTES_PER_LINE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_PALETTE_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_SCANNER_H_RES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_SCANNER_V_RES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCX_EXTRA	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCX_MAGIC	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCX_PAGE_LIST	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GEM_VERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GEM_HEADERSIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GEM_PLANES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GEM_PATTERNLENGTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GEM_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GEM_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_EPS_VERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_EPS_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_EPS_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_EPS_TITLE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_EPS_CREATOR	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_EPS_BOUNDINGBOX	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_EPS_TRANSLATE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_EPS_SCALE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_EPS_IMAGE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_WIDE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_HIGH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_XORG	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_YORG	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_PLANES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_MASK	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_COMPRESSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_TRAN_ASPT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_PAGE_W	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_PAGE_H	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_VIEW_MODE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_TRANSP_COLOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_X_ASPECT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IFF_Y_ASPECT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_MANUFACTURER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_VERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_IMAGETYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_HORZRES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_VERTRES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_BITSPERPIXEL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_PIXELSPERLINE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_STORAGEFMT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_TRANSFMT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_PREVPAGE	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_BTR_NEXTPAGE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_BTR_NUMLINES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IMT_TYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IMT_FMT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IMT_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IMT_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IMT_RESOLUTION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IMT_BITSWAP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IMT_SWAB	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_IMT_INVERT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_YORIGIN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_XORIGIN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_LINES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_PIXELS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_BITSPIX	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_COMPRESSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_BYTEFORMAT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_COMPVERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_YAXIS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_XAXIS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_NBLOCKTYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_DISPLAYMETHOD	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_XSEPERATION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_YSEPERATION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_BLOCKLENGTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LV_TEXT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_ICA_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_ICA_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_ICA_DEPTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_ICA_XDPI	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_ICA_YDPI	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_ICA_BITORDER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_ICA_BASE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_ICA_COMPRESSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_ICA_FILLORDER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_RAS_MAGIC	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_RAS_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_RAS_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_RAS_DEPTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_RAS_LENGTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_RAS_TYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_RAS_COLOR_MAP_TYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_RAS_COLOR_MAP_LENGTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_MAGIC	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_STORAGE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_BPC	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_DIMENSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_X_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_Y_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_Z_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_PIX_MIN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_PIX_MAX	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_DUMMY1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_IMAGE_NAME	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_COLOR_MAP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_SGI_DUMMY2	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_GIF_VERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_SCREEN_ASPECT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_SCREEN_BG_COLOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_SCREEN_FLAGS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_SCREEN_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_SCREEN_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_SCREEN_PALETTE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_IMAGE_LEFT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_IMAGE_TOP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_IMAGE_FLAGS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_IMAGE_PALETTE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_EXT_NUMBER_BEFORE_IMG	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_EXT_BEFORE_IMG	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_EXT_NUMBER_AFTER_IMG	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_EXT_AFTER_IMG	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_GIF_TRANSPARENT_COLOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_FH_KEY	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_FH_HANDLE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_FH_LEFT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_FH_TOP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_FH_RIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_FH_BOTTOM	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_FH_INCH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_FH_RESERVED	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_MH_FILE_TYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_MH_HEADER_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_MH_VERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_MH_FILE_SIZE	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_WMF_MH_NUM_OBJECTS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_MH_MAX_RECORD_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_WMF_MH_NO_PARAMETERS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CLP_FILE_ID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_CLP_FORMAT_COUNT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_KEY1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_KEY2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_X_AR_BITMAP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_Y_AR_BITMAP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_X_AR_PRINTER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_Y_AR_PRINTER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_X_PRINTER_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_Y_PRINTER_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_X_ASPECT_CORR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_Y_ASPECT_CORR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_CHECKSUM	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_MSP_PADDING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_ID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_HDR_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_HDR_VER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_IMAGE_ID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_LENGTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_FORMAT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_BIT_SEX	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_COLOR	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_KFX_XRES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_YRES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_PLANES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_BITS_PER_PIX	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_PAPER_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_DATE_CRT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_DATE_MOD	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_DATE_ACC	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_IDX_OFFSET	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_IDX_LEN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_COM_OFFSET	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_COM_LEN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_USER_OFFSET	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_USER_LEN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_DATA_OFFSET	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_KFX_DATA_LEN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_DATA_FORMAT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_OPTIONS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_ENCRYPTION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_AUTHENTICATION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_AUTH_MAC	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_DATA_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_REAL_BPP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_STORE_BPP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_REAL_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_STORE_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_REAL_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_STORE_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_NCR_ORIENT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_PMI	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_DATA_ENDIAN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_FILL_ORDER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_GRANULARITY	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_MIN_PIX_VALUE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_MAX_PIX_VALUE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_X_RES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_Y_RES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_RES_UNIT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_NCR_ERROR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_SIGNATURE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_VERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_ROWS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_COLS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_DEPTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_MODE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_MODE_LEN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_COMPRESSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_FILE_HDR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_COLOR_DATA	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_RESOURCE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_GLOBAL_MASK	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_RECT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_NUMBER_CHANNELS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_BLEND_MODE_KEY	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_OPACITY	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_PSD_LAYER_CLIPPING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_FLAGS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_MASK_DATA	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_ASCII_NAME	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_CHANNEL_LEN_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_BLENDING_RANGES_DATA	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_NUM_LAYERS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_GLOBAL_MASK_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_ADJUSTMENT_LAYER_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_UNICODE_NAME	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_LAYER_ID	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_EFFECT_LAYER_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_EFFECT_LAYER_COMMON_STATE_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_EFFECT_LAYER_SHADOW_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_EFFECT_LAYER_GLOW_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_EFFECT_LAYER_BEVEL_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_TOOL_TYPE_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_PATTERN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_ANNOTATIONS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_BLEND_CLIPPING_ELEMENTS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_BLEND_INTERIOR_ELEMENTS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_KNOCKOUT_SETTING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_PROTECTED_SETTING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_SHEET_COLOR_SETTING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_REFERENCE_POINT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_OBJ_BASED_EFFECTS_LAYER_INFO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PSD_GRADIENT_SETTINGS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_SIGNATURE	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_AFX_VER_MAJOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_VER_MINOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_VER_REV	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_VER_DEV	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_HDR_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_CHK_SUM	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_TYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_PRD_VER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_ENC_METHOD	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_COMMENT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_DATA_START	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_DATA_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_AFX_RES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPICA_RESERVED	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPICA_IMAGE_PACK_PARAMS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPICA_BASE4_STOP_OFFSET	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPICA_BASE16_STOP_OFFSET	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPICA_IPE_STOP_OFFSET	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPICA_IP_INTERLEAVE_RATIO	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_SIGNATURE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_VERSION_NUMBER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_SOFTWARE_RELEASE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_IMAGE_MAG_DESCRIPTION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_IMAGE_SCAN_TIME	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_LAST_MODIFICATION_DATE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_MED_ORIGINAL_RECORDING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_TYPE_ORIGINAL_RECORDING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_SCANNER_VENDOR	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_PCD_IPI_SCANNER_PRODUCT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_SCANNER_FIRMWARE_LEVEL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_SCANNER_FIRMWARE_DATE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_SCANNER_SERIAL_NUMBER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_SCANNER_PIXEL_SIZE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_EQUIPMENT_MANUFACTURER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_PHOTONAME_CHAR_SET	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_PHOTONAME_ESC_SEQ	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_PHOTONAME	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_SBA_DATA	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_COPYRIGHT_STATUS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCD_IPI_COPYRIGHT_FILENAME	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_LD_MASK_IMAGE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PCT_VERSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_IMAGEWIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_IMAGEHEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_BITSPERSAMPLE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_PHOTOMETRICINTERPRETATION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_SAMPLESPERPIXEL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_UNIQUECAMERAMODEL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_MAKE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_MODEL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_TIMESTAMP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_CFAREPEATPATTERNDIM	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_CFAPATTERN	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_BLACKLEVELREPEATDIM	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_BLACKLEVEL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_ASSHOTNEUTRAL	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_DCRAW_WHITELEVEL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_CALIBRATIONILLUMINANT1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_CALIBRATIONILLUMINANT2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_COLORMATRIX1	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_COLORMATRIX2	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_BASELINEEXPOSURE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_DCRAW_AS_SHOT_WHITEXY	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PTOCA_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_PTOCA_WIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_NEWSUBFILETYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_SUBFILETYPE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_IMAGEWIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_IMAGEHEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_BITSPERSAMPLE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_COMPRESSION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_PHOTOMETRICINTERPRETATION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_THRESHOLDING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_CELLWIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_CELLLENGTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_FILLORDER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_DOCUMENTNAME	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_IMAGEDESCRIPTION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_MAKE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_MODEL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_STRIPOFFSETS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_ORIENTATION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_SAMPLESPERPIXEL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_ROWSPERSTRIP	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_TIF_STRIPBYTECOUNTS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_MINSAMPLEVALUE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_MAXSAMPLEVALUE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_XRESNUMERATOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_XRESDENOMINATOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_YRESNUMERATOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_YRESDENOMINATOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_PLANARCONFIGURATION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_PAGENAME	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_XPOSITION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_YPOSITION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_FREEOFFSETS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_FREEBYTECOUNTS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_GRAYRESPONSEUNIT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_GRAYRESPONSECURVE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_T4OPTIONS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_T6OPTIONS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_RESOLUTIONUNIT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_PAGENUMBER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_COLORRESPONSEUNIT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_TRANSFERFUNCTION	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_SOFTWARE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_DATETIME	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_ARTIST	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_HOSTCOMPUTER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_PREDICTOR	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_WHITPOINT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_PRIMARYCHROMATICITIES	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_TIF_COLORMAP	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_HALFTONEHINTS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_TILEWIDTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_TILELENGTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_TILEOFFSETS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_TILEBYTECOUNTS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_BADFAXLINES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_CLEANFAXDATA	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_CONSECUTIVEBADFAXLINES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_INKSET	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_INKNAMES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_NUMBEROFINKS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_DOTRANGE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_TARGETPRINTER	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_EXTRASAMPLES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_SAMPLEFORMAT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_SMINSAMPLEVALUE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_SMAXSAMPLEVALUE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_TRANSFERRANGE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_JPEGPROC	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_JPEGINTERCHANGEFORMAT	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_JPEGINTERCHANGEFORMATLENGTH	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_JPEGRESTARTINTERVAL	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_JPEGLOSSLESSPREDICCTORS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_JPEGPOINTTRANSFORMS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_JPEGQTABLES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_JPEGDCTTABLES	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_JPEGACTTABLES	This value has been deprecated and will be removed from the public API in a future release.

IGTAG_TIF_YCBCRCOEFFICIENTS	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_YCBCRSUBSAMPLING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_YCBCRPOSITIONING	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_REFERENCEBLACKWHITE	This value has been deprecated and will be removed from the public API in a future release.
IGTAG_TIF_COPYRIGHT	This value has been deprecated and will be removed from the public API in a future release.

Remarks:

See Non-Image Data Processing for more details.

1.3.1.5.133 enumIGTGATagIDs

Lists all TGA tag identifiers.

Values:

IGMDTAG_ID_TGA_FORMAT	TGA metadata format identifier.
IGMDTAG_ID_TGA_HEAD	Header data section.
IGMDTAG_ID_TGA_HEAD_IDLENGTH	ID length.
IGMDTAG_ID_TGA_HEAD_COLORMAPTYPE	Color map type.
IGMDTAG_ID_TGA_HEAD_IMAGETYPE	Image type.
IGMDTAG_ID_TGA_HEAD_CMAPSTART	CMap start.
IGMDTAG_ID_TGA_HEAD_CMAPLENGTH	CMap length.
IGMDTAG_ID_TGA_HEAD_CMAPDEPTH	CMap depth.
IGMDTAG_ID_TGA_HEAD_XOFFSET	Absolute horizontal coordinate for the lower left corner of the image as it is positioned on a display device having an origin at the lower left of the screen.
IGMDTAG_ID_TGA_HEAD_YOFFSET	Absolute vertical coordinate for the lower left corner of the image as it is positioned on a display device having an origin at the lower left of the screen.
IGMDTAG_ID_TGA_HEAD_WIDTH	Image width.
IGMDTAG_ID_TGA_HEAD_HEIGHT	Image height.
IGMDTAG_ID_TGA_HEAD_PIXELDEPTH	Pixel depth.
IGMDTAG_ID_TGA_HEAD_IMAGEDESCRIPTOR	Image descriptor.
IGMDTAG_ID_TGA_FOOT	Footer data section.
IGMDTAG_ID_TGA_FOOT_EXTENSION_OFFSET	Extension offset.
IGMDTAG_ID_TGA_FOOT_DEVELOPER_OFFSET	Developer offset.
IGMDTAG_ID_TGA_FOOT_SIGNATURE	Signature string.
IGMDTAG_ID_TGA_EXT	Extension data section.
IGMDTAG_ID_TGA_EXT_SIZE	Extension size.
IGMDTAG_ID_TGA_EXT_AUTHORNAME	Author name.
IGMDTAG_ID_TGA_EXT_AUTHORCOMMENT	Author comment.
IGMDTAG_ID_TGA_EXT_STAMPMONTH	Stamp month.
IGMDTAG_ID_TGA_EXT_STAMPDAY	Stamp day.
IGMDTAG_ID_TGA_EXT_STAMPYEAR	Stamp year.
IGMDTAG_ID_TGA_EXT_STAMPHOUR	Stamp hour.
IGMDTAG_ID_TGA_EXT_STAMPMINUTE	Stamp minute.
IGMDTAG_ID_TGA_EXT_STAMPSECOND	Stamp second.
IGMDTAG_ID_TGA_EXT_JOBNAME	Job name .
IGMDTAG_ID_TGA_EXT_JOBHOUR	Job hour value.
IGMDTAG_ID_TGA_EXT_JOBMINUTE	Job minute value.
IGMDTAG_ID_TGA_EXT_JOBSECOND	Job second value.
IGMDTAG_ID_TGA_EXT_SOFTWAREID	Software ID.
IGMDTAG_ID_TGA_EXT_VERSIONNUMBER	Version number.
IGMDTAG_ID_TGA_EXT_VERSIONLETTER	Version letter.
IGMDTAG_ID_TGA_EXT_KEYCOLOR	Key color.
IGMDTAG_ID_TGA_EXT_PIXELNUMERATOR	Pixel numerator.
IGMDTAG_ID_TGA_EXT_PIXELDENOMINATOR	Pixel denominator.
IGMDTAG_ID_TGA_EXT_GAMMANUMERATOR	Gamma numerator.
IGMDTAG_ID_TGA_EXT_GAMMADENOMINATOR	Gamma denominator.

IGMDTAG_ID_TGA_EXT_COLOROFFSET	Color offset.
IGMDTAG_ID_TGA_EXT_STAMPOFFSET	Stamp offset.
IGMDTAG_ID_TGA_EXT_SCANOFFSET	Scan offset.
IGMDTAG_ID_TGA_EXT_ATTRIBUTESTYPE	Attributes type.
IGMDTAG_ID_TGA_IMAGE_ID	Image ID tag.

1.3.1.5.134 enumIGTIFFTagIDs

Lists all TIF tag identifiers. See TIFF 6.0, TIFF/EP, DNG specifications for more details.

Values:

IGMDTAG_ID_TIF_FORMAT	TIF metadata format identifier.
IGMDTAG_ID_TIF_NEW_SUBFILE_TYPE	New subfile type.
IGMDTAG_ID_TIF_SUBFILE_TYPE	Subfile type.
IGMDTAG_ID_TIF_IMAGE_WIDTH	Image width.
IGMDTAG_ID_TIF_IMAGE_HEIGHT	Image height.
IGMDTAG_ID_TIF_BITS_PER_SAMPLE	Bits per sample.
IGMDTAG_ID_TIF_COMPRESSION	Image data compression type.
IGMDTAG_ID_TIF_PHOTO_INTERP	Photometric interpretation.
IGMDTAG_ID_TIF_THRESHOLDING	The technique used to convert from gray to black and white pixels.
IGMDTAG_ID_TIF_CELL_WIDTH	Cell width.
IGMDTAG_ID_TIF_CELL_HEIGHT	Cell height.
IGMDTAG_ID_TIF_FILL_ORDER	Fill order.
IGMDTAG_ID_TIF_DOC_NAME	Document name.
IGMDTAG_ID_TIF_DESCRIPTION	Image description.
IGMDTAG_ID_TIF_MAKE	Manufacturer of equipment used to generate the image.
IGMDTAG_ID_TIF_MODEL	The model name of equipment used to generate the image.
IGMDTAG_ID_TIF_STRIP_OFFSETS	Strip offsets.
IGMDTAG_ID_TIF_ORIENTATION	Image orientation.
IGMDTAG_ID_TIF_SAMPLES_PER_PIXEL	Samples per pixel.
IGMDTAG_ID_TIF_ROWS_PER_STRIP	Rows per strip.
IGMDTAG_ID_TIF_STRIP_BYTE_COUNT	Strip byte counts.
IGMDTAG_ID_TIF_MIN_SAMPLE_VAL	Min sample value.
IGMDTAG_ID_TIF_MAX_SAMPLE_VAL	Max sample value.
IGMDTAG_ID_TIF_X_RES	X resolution.
IGMDTAG_ID_TIF_Y_RES	Y resolution.
IGMDTAG_ID_TIF_PLANAR_CONFIG	Planar configuration.
IGMDTAG_ID_TIF_PAGE_NAME	Page name.
IGMDTAG_ID_TIF_X_POS	X position.
IGMDTAG_ID_TIF_Y_POS	Y position.
IGMDTAG_ID_TIF_FREE_OFFSETS	Free offsets.
IGMDTAG_ID_TIF_FREE_BYTE_COUNTS	Free byte counts.
IGMDTAG_ID_TIF_GRAY_RESPONSE_UNIT	Gray response unit.
IGMDTAG_ID_TIF_GRAY_RESPONSE_CURVE	Gray response curve.
IGMDTAG_ID_TIF_T4_OPTIONS	T4 options.
IGMDTAG_ID_TIF_T6_OPTIONS	T6 options.
IGMDTAG_ID_TIF_RES_UNIT	Resolution unit.
IGMDTAG_ID_TIF_PAGE_NUMBER	Page number.
IGMDTAG_ID_TIF_TRANSFER_FUNC	Transfer function.
IGMDTAG_ID_TIF_SOFTWARE	Name and version number of the software package(s) used to create the image.
IGMDTAG_ID_TIF_DATE_TIME	Date time.
IGMDTAG_ID_TIF_ARTIST	Person who created the image.

IGMDTAG_ID_TIF_HOST_COMPUTER	Host computer.
IGMDTAG_ID_TIF_PREDICTOR	Mathematical operator that is applied to the image data before an encoding scheme is applied.
IGMDTAG_ID_TIF_WHITE_POINT	White point.
IGMDTAG_ID_TIF_PRIMARY_CHROMA	Primary chromaticities.
IGMDTAG_ID_TIF_COLOR_MAP	Color map.
IGMDTAG_ID_TIF_HALFTONE_HINTS	Halftone hints.
IGMDTAG_ID_TIF_TILE_WIDTH	Tile width.
IGMDTAG_ID_TIF_TILE_HEIGHT	Tile height.
IGMDTAG_ID_TIF_TILE_OFFSETS	Tile offsets.
IGMDTAG_ID_TIF_TILE_BYTE_COUNT	Tile byte counts.
IGMDTAG_ID_TIF_SUBIFDS	Child IFDs offsets.
IGMDTAG_ID_TIF_INK_SET	The set of inks used.
IGMDTAG_ID_TIF_INK_NAMES	Ink names.
IGMDTAG_ID_TIF_NUMBER_OF_LINKS	Number of inks.
IGMDTAG_ID_TIF_DOT_RANGE	Dot range.
IGMDTAG_ID_TIF_TARGET_PRINTER	Target printer.
IGMDTAG_ID_TIF_EXTRA_SAMPLES	Extra samples.
IGMDTAG_ID_TIF_SAMPLE_FORMAT	Sample format.
IGMDTAG_ID_TIF_SMIN_SAMPLE_VAL	S min sample value.
IGMDTAG_ID_TIF_SMAX_SAMPLE_VAL	S max sample value.
IGMDTAG_ID_TIF_TRANSFER_RANGE	Transfer range.
IGMDTAG_ID_TIF_JPEG_TABLES	JPEG tables.
IGMDTAG_ID_TIF_JPEG_PROC	JPEG proc.
IGMDTAG_ID_TIF_JPEG_INTERCHANGE	JPEG interchange format.
IGMDTAG_ID_TIF_JPEG_INTERCHANGE_LEN	JPEG interchange format length.
IGMDTAG_ID_TIF_JPEG_RESTART_INTERVAL	JPEG restart interval.
IGMDTAG_ID_TIF_JPEG_LOSSLESS_PREDICTOR	JPEG lossless predictors.
IGMDTAG_ID_TIF_JPEG_POINT_TRANSFORMS	JPEG point transforms.
IGMDTAG_ID_TIF_JPEG_Q_TABLES	JPEG Q tables.
IGMDTAG_ID_TIF_JPEG_DC_TABLES	JPEG DC tables.
IGMDTAG_ID_TIF_JPEG_AC_TABLES	JPEG AC tables.
IGMDTAG_ID_TIF_YCBCR_COEFFICIENTS	YCbCr coefficients.
IGMDTAG_ID_TIF_YCBCR_SUBSAMPLING	YCbCr sub sampling.
IGMDTAG_ID_TIF_YCBCR_POS	YCbCr positioning.
IGMDTAG_ID_TIF_REFERENCE_BLACK_WHITE	Reference black white.
IGMDTAG_ID_TIF_XMP_METADATA	XML packet containing XMP metadata.
IGMDTAG_ID_TIF_RATING	Image rating. Valid values are from 0 to 5.
IGMDTAG_ID_TIF_MICROSOFT_PHOTO_RATING	Microsoft photo rating. Valid values are from 0 to 99.
IGMDTAG_ID_TIF_CFA_REPEAT_PATTERN_DIM	CFA repeat pattern dim.
IGMDTAG_ID_TIF_CFA_PATTERN	CFA pattern.
IGMDTAG_ID_TIF_BATTERY_LEVEL	Battery level.
IGMDTAG_ID_TIF_COPYRIGHT	Copyright notice.
IGMDTAG_ID_TIF_EXPOSURE_TIME	Exposure time.
IGMDTAG_ID_TIF_FNUMBER	Actual lens f-number used when the image was captured.
IGMDTAG_ID_TIF_IPTC_NAA	IPTC / NAA.
IGMDTAG_ID_TIF_PHOTOSHOP_RESOURCES	Photoshop resources.
IGMDTAG_ID_TIF_EXIF_IFDPOINTER	Exif IFD pointer.

IGMDTAG_ID_TIF_ICC_PROFILE	ICC color profile.
IGMDTAG_ID_TIF_EXPOSURE_PROGRAM	Exposure program.
IGMDTAG_ID_TIF_SPECTRAL_SENSITIVITY	Spectral sensitivity.
IGMDTAG_ID_TIF_GPSINFOIFDPOINTER	Exif IFD pointer.
IGMDTAG_ID_TIF_ISOSPEEDRAITINGS	ISO speed raitings.
IGMDTAG_ID_TIF_OECF	Indicates the Opto-Electric Conversion Function (OECF) specified in ISO 14524.
IGMDTAG_ID_TIF_INTERLACE	Indicates the field number of multifield images.
IGMDTAG_ID_TIF_TIMEZONE_OFFSET	Time zone offset.
IGMDTAG_ID_TIF_SELFTIMER_MODE	Self timer mode.
IGMDTAG_ID_TIF_DATETIMEORIGINAL	Date time original.
IGMDTAG_ID_TIF_COMPRESSEDBITSPERPIXEL	Compressed bits per pixel.
IGMDTAG_ID_TIF_SHUTTERSPEED_VALUE	Shutter speed value.
IGMDTAG_ID_TIF_APERTURE_VALUE	Aperture value.
IGMDTAG_ID_TIF_BRIGHTNESS_VALUE	Brightness value.
IGMDTAG_ID_TIF_EXPOSURE_BIAS_VALUE	Exposure bias value.
IGMDTAG_ID_TIF_MAXAPERTURE_VALUE	Max aperture value.
IGMDTAG_ID_TIF_SUBJECTDISTANCE	Subject distance.
IGMDTAG_ID_TIF_METERING_MODE	Metering mode.
IGMDTAG_ID_TIF_LIGHT_SOURCE	Light source.
IGMDTAG_ID_TIF_FLASH	Indicates weither or not flash used when the image was captured.
IGMDTAG_ID_TIF_FOCAL_LENGTH	Focal length.
IGMDTAG_ID_TIF_FLASHENERGY	Flash energy.
IGMDTAG_ID_TIF_SPATIAL_FREQUENCY_RESPONSE	Spatial frequency response.
IGMDTAG_ID_TIF_NOISE	Noise measurement values.
IGMDTAG_ID_TIF_FOCAL_PLANE_XRESOLUTION	Focal plane xresolution.
IGMDTAG_ID_TIF_FOCAL_PLANE_YRESOLUTION	Focal plane yresolution.
IGMDTAG_ID_TIF_FOCAL_PLANE_RESOLUTION_UNIT	Focal plane resolution unit.
IGMDTAG_ID_TIF_IMAGE_NUMBER	Image number.
IGMDTAG_ID_TIF_SECURITY_CLASSIFICATION	Security classification.
IGMDTAG_ID_TIF_IMAGE_HISTORY	Image history.
IGMDTAG_ID_TIF_SUBJECT_LOCATION	Subject location.
IGMDTAG_ID_TIF_EXPOSURE_INDEX	Exposure index.
IGMDTAG_ID_TIF_TIFEPS_STANDARDID	TIFF/EP standard ID.
IGMDTAG_ID_TIF_SENSING_METHOD	Sensing method.
IGMDTAG_ID_TIF_DNG_VERSION	DNG version.
IGMDTAG_ID_TIF_DNG_BACKWARdVERSION	DNG backward version.
IGMDTAG_ID_TIF_UNIQUE_CAMERAMODEL	Unique camera model.
IGMDTAG_ID_TIF_LOCALIZED_CAMERAMODEL	Localized camera model.
IGMDTAG_ID_TIF_CFA_PlANECOLOR	CFA plane color.
IGMDTAG_ID_TIF_CFA_LAYOUT	CFA layout.
IGMDTAG_ID_TIF_LINEARIZATION_TABLE	Linearization table.
IGMDTAG_ID_TIF_BLACK_LEVELREPEAT_DIM	Black level repeat dim.
IGMDTAG_ID_TIF_BLACK_LEVEL	Black level.
IGMDTAG_ID_TIF_BLACK_LEVEL_DELTAH	Black level delta H.
IGMDTAG_ID_TIF_BLACK_LEVEL_DELTAV	Black level delta V.
IGMDTAG_ID_TIF_WHITE_LEVEL	White level.

IGMDTAG_ID_TIF_DEFAULT_SCALE	Default scale.
IGMDTAG_ID_TIF_BEST_QUALITY_SCALE	Best quality scale.
IGMDTAG_ID_TIF_DEFAULT_CROP_ORIGIN	Default crop origin.
IGMDTAG_ID_TIF_DEFAULT_CROP_SIZE	Default crop size.
IGMDTAG_ID_TIF_CALIBRATION_ILLUMINANT1	Calibration illuminant 1.
IGMDTAG_ID_TIF_CALIBRATION_ILLUMINANT2	Calibration illuminant 2.
IGMDTAG_ID_TIF_COLOR_MATRIX1	Color matrix 1.
IGMDTAG_ID_TIF_COLOR_MATRIX2	Color matrix 2.
IGMDTAG_ID_TIF_CAMERA_CALIBRATION1	Camera calibration 1.
IGMDTAG_ID_TIF_CAMERA_CALIBRATION2	Camera calibration 2.
IGMDTAG_ID_TIF_REDUCTION_MATRIX1	Reduction matrix 1.
IGMDTAG_ID_TIF_REDUCTION_MATRIX2	Reduction matrix 2.
IGMDTAG_ID_TIF_ANALOG_BALANCE	Analog balance.
IGMDTAG_ID_TIF_AS_SHOT_NEUTRAL	As shot neutral.
IGMDTAG_ID_TIF_AS_SHOT_WHITEXY	As shot white XY.
IGMDTAG_ID_TIF_BASELINE_EXPOSURE	Baseline exposure.
IGMDTAG_ID_TIF_BASELINE_NOISE	Baseline noise.
IGMDTAG_ID_TIF_BASELINE_SHARPNESS	Baseline sharpness.
IGMDTAG_ID_TIF_BAYER_GREEN_SPLIT	Bayer green split.
IGMDTAG_ID_TIF_LINEAR_RESPONSE_LIMIT	Linear response limit.
IGMDTAG_ID_TIF_CAMERA_SERIAL_NUMBER	Camera serial number.
IGMDTAG_ID_TIF_LENS_INFO	Lens info.
IGMDTAG_ID_TIF_CHROMA_BLUR_RADIUS	Chroma blur radius.
IGMDTAG_ID_TIF_ANTI_ALIAS_STRENGTH	Anti alias strength.
IGMDTAG_ID_TIF_DNG_PRIVATE_DATA	DNG private data.
IGMDTAG_ID_TIF_MAKER_NOTE_SAFETY	Maker note safety.
IGMDTAG_ID_TIF_SHADOW_SCALE	Shadow scale.
IGMDTAG_ID_TIF_RAW_DATA_UNIQUE_ID	Raw data unique ID.
IGMDTAG_ID_TIF_ORIGINAL_RAW_FILE_NAME	Original raw file name.
IGMDTAG_ID_TIF_ORIGINAL_RAW_FILE_DATA	Original raw file data.
IGMDTAG_ID_TIF_ACTIVE_AREA	Active area.
IGMDTAG_ID_TIF_MASKED_AREAS	Masked areas.
IGMDTAG_ID_TIF_ASSHOT_ICC_PROFILE	As shot ICC profile.
IGMDTAG_ID_TIF_ASSHOT_PRE_PROFILE_MATRIX	As shot pre profile matrix.
IGMDTAG_ID_TIF_CURRENT_ICC_PROFILE	Current ICC profile.
IGMDTAG_ID_TIF_CURRENT_PRE_PROFILE_MATRIX	Current pre profile matrix.
IGMDTAG_ID_TIF_HEADER	For internal use only.
IGMDTAG_ID_TIF_JPEG_INTERCHANGE_DATA	For internal use only.

1.3.1.5.135 enumIGTwistModes

This enumeration contains rotation modes for IG_FX_twist function.

Values:

IG_TWIST_90	90 degrees.
IG_TWIST_180	180 degrees.
IG_TWIST_270	270 degrees.
IG_TWIST_RANDOM	Random.

1.3.1.5.136 enumIGTypeIDs

Specifies ImageGear data type IDs.

Values:

AM_TID_VOID	Data type is AT_VOID.
AM_TID_CHAR	Data type is AT_CHAR.
AM_TID_BYTE	Data type is AT_BYTE.
AM_TID_SHORT	Data type is SHORT (AT_INT16).
AM_TID_WORD	Data type is AT_WORD.
AM_TID_INT	Data type is INT (AT_INT32).
AM_TID_UINT	Data type is UINT (AT_UINT32).
AM_TID_LONG	Data type is LONG (AT_INT32).
AM_TID_DWORD	Data type is AT_DWORD.
AM_TID_AT_MODE	Data type is AT_MODE.
AM_TID_AT_BOOL	Data type is AT_BOOL.
AM_TID_AT_LMODE	Data type is AT_LMODE.
AM_TID_AT_DIMENSION	Data type is AT_DIMENSION.
AM_TID_AT_RECT	Data type is AT_RECT.
AM_TID_DOUBLE	Data type is AT_DOUBLE.
AM_TID_RGBQUAD	Data type is AT_RGBQUAD.
AM_TID_STRING	Data type is NULL-terminating array of AT_CHAR.
AM_TID_FLOAT	Data type is AT_FLOAT.
AM_TID_LP	Data type modifier to describe pointer to data.
AM_TID_TMASK	Bit mask to extract data type from type description.
AM_TID_PMASK	Bit mask to extract data type modifier from type description.
AM_TID_PSHIFT	Bit offset of data type modifier in type description.

1.3.1.5.137 enumIGWBMPTagIDs

Lists all WBMP tag identifiers.

Values:

IGMDTAG_ID_WBMP_FORMAT	WBMP metadata format identifier.

1.3.1.5.138 enumIGWipeStyles

Identifies the different image transition effects.

Values:

IG_WIPE_LEFTTORIGHT	Left-to-Right Wipe.
IG_WIPE_RIGHTTOLEFT	Right-To-Left Wipe.
IG_WIPE_UP_TO_DOWN	Up-to-Down Wipe.
IG_WIPE_DOWN_TO_UP	Down-to-Up Wipe.
IG_WIPE_SPARKLE	Sparkle Transition.
IG_WIPE_ULTOLRDIAG	Upper Left to Lower Right wipe.
IG_WIPE_LRTOULDIAG	Lower Right to Upper Left wipe.
IG_WIPE_URTOLLDIAG	Upper Right to Lower Left wipe.
IG_WIPE_LLTOURDIAG	Lower Left to Upper Right wipe.
IG_WIPE_CLOCK	Clockwise wipe.
IG_WIPE_SPARKLE_CLOCK	Clockwise wipe with sparkles.
IG_WIPE_DOUBLE_CLOCK	Two simultaneous clockwise wipes, 180 grades apart.
IG_WIPE_SLIDE_RIGHT	New image slides in from the left.
IG_WIPE_SLIDE_LEFT	New image slides in from the right.
IG_WIPE_SLIDE_UP	New image slides in from the down.
IG_WIPE_SLIDE_DOWN	New image slides in from the up.
IG_WIPE_RANDOM_BARS_DOWN	Vertical bars of old image fall to reveal new image.
IG_WIPE_RAIN	Vertical lines of new image cover over old, like paint running down the side of a bucket.
IG_WIPE_BOOK	Book wipe.
IG_WIPE_ROLL	Old image rolls in from right to left.
IG_WIPE_UNROLL	New image rolls out from left to right.
IG_WIPE_EXPAND_PROPORTIONAL	New image expands from the center of old image in diagonal directions.
IG_WIPE_EXPAND_HORIZONTAL	New image expands from the center of old image in horizontal directions.
IG_WIPE_EXPAND_VERTICAL	New image expands from the center of old image in vertical directions.
IG_WIPE_STRIPS_HORIZONTAL	New image appears as expanding horizontal strips.
IG_WIPE_STRIPS_VERTICAL	New image appears as expanding vertical strips.
IG_WIPE_CELLS	New image appears as expanding square cells.
IG_WIPE_BALL	New image appears as tracks of spirally moving balls.
IG_WIPE_GEARS	New image appears as tracks of moving ImageGear's icons.

1.3.1.5.139 enumIGWMFTagIDs

Lists all WMF tag identifiers.

Values:

IGMDTAG_ID_WMF_FORMAT	WMF metadata format identifier.
IGMDTAG_ID_WMF_FH_KEY	WMF file magic number.
IGMDTAG_ID_WMF_FH_HANDLE	Metafile HANDLE number (should always be 0).
IGMDTAG_ID_WMF_FH_LEFT	Left coordinate in metafile units.
IGMDTAG_ID_WMF_FH_TOP	Top coordinate in metafile units.
IGMDTAG_ID_WMF_FH_RIGHT	Right coordinate in metafile units.
IGMDTAG_ID_WMF_FH_BOTTOM	Bottom coordinate in metafile units..
IGMDTAG_ID_WMF_FH_INCH	Number of metafile units per inch.
IGMDTAG_ID_WMF_FH_RESERVED	Reserved (should always be 0).
IGMDTAG_ID_WMF_MH_FILE_TYPE	Type of metafile (1=memory, 2=disk).
IGMDTAG_ID_WMF_MH_HEADER_SIZE	Size of header in WORDS (always 9).
IGMDTAG_ID_WMF_MH_VERSION	Version of Microsoft Windows used.
IGMDTAG_ID_WMF_MH_FILE_SIZE	Total size of the metafile in WORDs.
IGMDTAG_ID_WMF_MH_NUM_OBJECTS	Number of objects in the file.
IGMDTAG_ID_WMF_MH_MAX_RECORD_SIZE	The size of largest record in WORDs.
IGMDTAG_ID_WMF_MH_NO_PARAMETERS	Not Used (always 0).

1.3.1.5.140 enumIGWPGTagIDs

Lists all WPG tag identifiers.

Values:

IGMDTAG_ID_WPG_FORMAT	WPG metadata format identifier.

1.3.1.5.141 enumIGXBMTagIDs

Lists all XBM tag identifiers.

Values:

IGMDTAG_ID_XBM_FORMAT	XBM metadata format identifier.

1.3.1.5.142 enumIGXMPTagIDs

Lists all XMP tag identifiers.

Values:

IGMDTAG_ID_XMP_FORMAT	XMP Metadata Format identifier.
IGMDTAG_ID_XMP_DESCRIPTION	XMP Schema tree.
IGMDTAG_ID_XMP_NAMESPACE	Namespace tree.
IGMDTAG_ID_XMP_PREFIX	Namespace prefix value.
IGMDTAG_ID_XMP_URI	Namespace URI value.
IGMDTAG_ID_XMP_ABOUT	About attribute value.
IGMDTAG_ID_XMP_PROPERTIES	Properties tree.
IGMDTAG_ID_XMP_PROPERTY	Property tree.
IGMDTAG_ID_XMP_PROPERTY_VALUE	Property value.
IGMDTAG_ID_XMP_PROPERTY_LANG	Language alternative tree.
IGMDTAG_ID_XMP_PROPERTY_QUA	Qualifiers tree.
IGMDTAG_ID_XMP_PROPERTY_BAG	Bag of values (unordered array) tree.
IGMDTAG_ID_XMP_PROPERTY_ALT	Alternative array of values tree.
IGMDTAG_ID_XMP_PROPERTY_SEQ	Sequence of values (ordered array) tree.
IGMDTAG_ID_XMP_PROPERTY_STRUCT	Structure tree.

Remarks:

These identifiers represent structural types of XMP metadata, such as Value, Sequence, Bag, Qualifier, etc. ImageGear does not provide enumerations for particular properties of XMP schemes. For more information about XMP metadata support, see Working with XMP Metadata.

1.3.1.5.143 enumIGXPMTagIDs

Lists all XPM tag identifiers.

Values:

IGMDTAG_ID_XPM_FORMAT	XPM metadata format identifier.

1.3.1.5.144 enumIGXWDTagIDs

Lists all XWD tag identifiers.

Values:

IGMDTAG_ID_XWD_FORMAT	XWD metadata format identifier.
IGMDTAG_ID_XWD_HEADER_SIZE	Header size. R/O.
IGMDTAG_ID_XWD_FILE_VERSION	File version. R/O.
IGMDTAG_ID_XWD_PIXMAP_FORMAT	Pixmap format. R/O.
IGMDTAG_ID_XWD_PIXMAP_DEPTH	Pixmap depth. R/O.
IGMDTAG_ID_XWD_PIXMAP_WIDTH	Pixmap width. R/O.
IGMDTAG_ID_XWD_PIXMAP_HEIGHT	Pixmap height. R/O.
IGMDTAG_ID_XWD_X_OFFSET	Bitmap x offset. R/W.
IGMDTAG_ID_XWD_BYTE_ORDER	Byte order. R/O.
IGMDTAG_ID_XWD_BITMAP_UNIT	Bitmap unit. R/O.
IGMDTAG_ID_XWD_BITMAP_BIT_ORDER	Bitmap bit order (MSBFirst, LSBFirst). R/O.
IGMDTAG_ID_XWD_BITMAP_PAD	Bitmap scanline pad. R/O.
IGMDTAG_ID_XWD_BITS_PER_PIXEL	Bits per pixel. R/O.
IGMDTAG_ID_XWD_BYTES_PER_LINE	Bytes per scanline. R/O.
IGMDTAG_ID_XWD_VISUAL_CLASS	Class of colormap. R/O.
IGMDTAG_ID_XWD_RED_MASK	Red mask. R/O.
IGMDTAG_ID_XWD_GREEN_MASK	Green mask. R/O.
IGMDTAG_ID_XWD_BLUE_MASK	Blue mask. R/O.
IGMDTAG_ID_XWD_BITS_PER_RGB	Log2 of distinct color values. R/O.
IGMDTAG_ID_XWD_NUMBER_OF_COLORS	Colors number. R/O.
IGMDTAG_ID_XWD_COLOR_MAP_ENTRIES	Color map entries. R/O.
IGMDTAG_ID_XWD_WINDOW_WIDTH	Window width. R/W.
IGMDTAG_ID_XWD_WINDOW_HEIGHT	Window height. R/W.
IGMDTAG_ID_XWD_WINDOW_X	Window upper left X coordinate. R/W.
IGMDTAG_ID_XWD_WINDOW_Y	Window upper left Y coordinate. R/W.
IGMDTAG_ID_XWD_WINDOW_BORDER_WIDTH	Window border width. R/W.

1.3.1.5.145 enumJPG_DCM

Specifies JPEG decimation types.

Values:

IG_JPG_DCM_1x1_1x1_1x1	Decimation value 1x1_1x1_1x1.
IG_JPG_DCM_2x1_1x1_1x1	Decimation value 2x1_1x1_1x1.
IG_JPG_DCM_1x2_1x1_1x1	Decimation value 1x2_1x1_1x1.
IG_JPG_DCM_2x2_1x1_1x1	Decimation value 2x2_1x1_1x1.
IG_JPG_DCM_2x2_2x1_2x1	Decimation value 2x2_2x1_2x1.
IG_JPG_DCM_4x2_1x1_1x1	Decimation value 4x2_1x1_1x1.
IG_JPG_DCM_2x4_1x1_1x1	Decimation value 2x4_1x1_1x1.
IG_JPG_DCM_4x1_1x1_1x1	Decimation value 4x1_1x1_1x1.
IG_JPG_DCM_1x4_1x1_1x1	Decimation value 1x4_1x1_1x1.
IG_JPG_DCM_4x1_2x1_2x1	Decimation value 4x1_2x1_2x1.
IG_JPG_DCM_1x4_1x2_1x2	Decimation value 1x4_1x2_1x2.
IG_JPG_DCM_4x4_2x2_2x2	Decimation value 4x4_2x2_2x2.

Remarks:

The format of these ImageGear decimation constants is: IG_JPG_DCM_<H1>x<V1>_<H2>x<V2>_<H3>x<V3>, where Hi, Vi = horizontal and vertical decimation values for the i-channel. For a more detailed definition, see the JPEG Specification.

1.3.1.5.146 enumLayoutConstants

Specifies bit flags indicating which arguments should be taken into account by IG_dspl_layout_set function.

Values:

IG_DSPL_IMAGE_RECT	Indicates that value of lpImageRect parameter of function should be taken into account.
IG_DSPL_DEVICE_RECT	Indicates that value of lpDeviceRect parameter of function should be taken into account.
IG_DSPL_CLIP_RECT	Indicates that value of lpClipRect parameter of function should be taken into account.
IG_DSPL_FIT_MODE	Indicates that value of nFitMode parameter of function should be taken into account.
IG_DSPL_ALIGN_MODE	Indicates that value of nAlignMode parameter of function should be taken into account.
IG_DSPL_ASPECT_MODE	Indicates that value of nAspectMode parameter of function should be taken into account.
IG_DSPL_ASPECT_VALUE	Indicates that value of dblAspectValue parameter of function should be taken into account.

Remarks:

See IG_dspl_layout_set for more details.

1.3.1.5.147 enumLoadColor

Specifies color reduction modes on image loading.

Values:

IG_LOAD_COLOR_DEFAULT	No color reduction performed.
IG_LOAD_COLOR_1	Image bit depth is reduced to 1 bit per pixel during loading.
IG_LOAD_COLOR_4	Image bit depth is reduced to 4 bits per pixel during loading.
IG_LOAD_COLOR_8	Image bit depth is reduced to 8 bits per pixel during loading.
IG_LOAD_GRAYSCALE_8	Image bit depth is reduced to 8 bits per pixel and color space converted to grayscale during loading.

Remarks:

See IG_load_color_reduction_set for more details.

1.3.1.5.148 enumLoadDoc

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_LOADDOC_DISPLAY_FIRST	This value has been deprecated and will be removed from the public API in a future release.
IG_LOADDOC_DISPLAY_ALL	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.149 enumMaxKern

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_MAX_KERN_HEIGHT	This value has been deprecated and will be removed from the public API in a future release.
IG_MAX_KERN_WIDTH	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.150 enumMPAppend

Multi-page image Append flag.

Values:

IG_APPEND_PAGE	This value is used as page number in image saving functions. It specifies that the page is to be appended to the multi-page image file.

1.3.1.5.151 enumMPCBMODE_MPI

Specifies notification codes for multi-page image operations.

Values:

IG_MPCBMODE_MPI_DELETE	Notify application that multi-page image is going to be deleted.
IG_MPCBMODE_MPI_ASSOCIATED	Notify application that multi-page image has been associated with external file or memory image.
IG_MPCBMODE_MPI_CLOSE	Notify application that multi-page image is going to close associated external file or memory image.
IG_MPCBMODE_MPI_CB_SET	Notify application that this callback data just has been set. Only the callback function that just has been set receives this notification.
IG_MPCBMODE_MPI_CB_RESET	Notify application that this callback data is to be reset.
IG_MPCBMODE_MPI_PAGEINSERTED	Application inserted new pages into multi-page image.
IG_MPCBMODE_MPI_PAGEUPDATED	Application updated pages in the multi-page image.
IG_MPCBMODE_MPI_PAGEDELETED	Application deleted pages in the multi-page image.
IG_MPCBMODE_MPF_PAGEINSERTED	Application inserted new pages into external file image.
IG_MPCBMODE_MPF_PAGEUPDATED	Application updated pages in the external multi-page image file.
IG_MPCBMODE_MPF_PAGEDELETED	Application deleted pages in the external multi-page image file.

Remarks:

See IG_mpi_CB_set for more details.

1.3.1.5.152 enumOrientation

Specifies image orientation units.

Values:

IG_ORIENT_TOP_LEFT	Image orientation is Row0=Top, Col0=Left (normal / portrait).
IG_ORIENT_TOP_RIGHT	Image orientation is Row0=Top, Col0=Right (flipped horizontally).
IG_ORIENT_BOTTOM_RIGHT	Image orientation is Row0=Bottom, Col0=Right (rotated by 180 degrees).
IG_ORIENT_BOTTOM_LEFT	Image orientation is Row0=Bottom, Col0=Left (flipped vertically).
IG_ORIENT_LEFT_TOP	Image orientation is Row0=Left, Col0=Top (rotated by 90 degrees counterclockwise and then flipped vertically).
IG_ORIENT_RIGHT_TOP	Image orientation is Row0=Right, Col0=Top (rotated by 90 degrees clockwise / landscape).
IG_ORIENT_RIGHT_BOTTOM	Image orientation is Row0=Right, Col0=Bottom (rotated by 90 degrees clockwise and then flipped vertically).
IG_ORIENT_LEFT_BOTTOM	Image orientation is Row0=Left, Col0=Bottom (rotated by 90 degrees counterclockwise / landscape).

Remarks:

There are 8 possible orientations. This enum labels them according to where the first row (row 0) and first col (col 0) of the image data is to be displayed. Regular images are displayed with row 0 at the top and column 0 at the left. This corresponds to IG_ORIENT_TOP_LEFT mode. The other orientations are combinations of flips and rotates. Portrait is usually IG_ORIENT_TOP_LEFT, and Landscape is either IG_ORIENT_RIGHT_TOP or IG_ORIENT_LEFT_BOTTOM.

1.3.1.5.153 enumPDFSaveFlags

Specifies control parameters for PDF image saving.

Values:

IG_PDF_DONT_SAVE_FILE_ATTRIBUTES	Prevents the file attributes and security settings of a PDF document opened from an existing PDF file from being copied over when saved to a new PDF file.
IG_PDF_LINEARIZED	Writes the file linearized for page serving over the remote connections.
IG_PDF_OPTIMIZE_XOBJECTS	Merges identical forms and images, as determined by an MD5 hash of their contents.
IG_PDF_OPTIMIZED	Performs garbage collection on unreferenced objects.

1.3.1.5.154 enumPDFTextEnc

Specifies the encoding scheme to be used to convert binary image data to the text format when saving raster image into the PDF document. Used with PDF filter TEXT_ENCODING control parameter.

Values:

IG_PDF_TEXTENC_NONE	Specifies that no encoding will be used.
IG_PDF_TEXTENC_ASCII_85	Specifies that ASCII 85 encoding will be used.
IG_PDF_TEXTENC_ASCII_HEX	Specifies that ASCII HEX encoding will be used.

1.3.1.5.155 enumPixdumpComponent

This enumeration has been deprecated and will be removed from the public API in a future release. Please use enumPixdumpComponentEx instead.

Values:

IG_GUI_PIXDUMP_COMPONENT_R	This value has been deprecated and will be removed from the public API in a future release.
IG_GUI_PIXDUMP_COMPONENT_G	This value has been deprecated and will be removed from the public API in a future release.
IG_GUI_PIXDUMP_COMPONENT_B	This value has been deprecated and will be removed from the public API in a future release.
IG_GUI_PIXDUMP_COMPONENT_RGB	This value has been deprecated and will be removed from the public API in a future release.
IG_GUI_PIXDUMP_COMPONENT_I	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.156 enumPixdumpComponentEx

GUI pixel dump window color components.

Values:

IG_GUI_PIXDUMP_COMPONENT_1	Display value of Component 1 of image pixels.
IG_GUI_PIXDUMP_COMPONENT_2	Display value of Component 2 of image pixels.
IG_GUI_PIXDUMP_COMPONENT_3	Display value of Component 3 of image pixels.
IG_GUI_PIXDUMP_COMPONENT_4	Display value of Component 4 of image pixels.
IG_GUI_PIXDUMP_COMPONENT_ALPHA	Display value of Alpha channel of image pixels.
IG_GUI_PIXDUMP_COMPONENT_EXTRA	Display value of Extra channels of image pixels.
IG_GUI_PIXDUMP_COMPONENT_ALL	Display value of all color components of image pixels.

Remarks:

Specifies color components to display. Color components indices are 1-based i.e. first component is IG_GUI_PIXDUMP_COMPONENT_1 and so on. See IG_GUI_pixdump_attribute_set for more details.

1.3.1.5.157 enumPixdumpData

GUI pixel dump window attributes.

Values:

IG_GUI_PIXDUMP_FONT	Specifies HFONT font handle used to display content of window.
IG_GUI_PIXDUMP_MODE	Specifies the mode to data display. Attribute value is a combination of enumPixdumpMode constants.
IG_GUI_PIXDUMP_COLOR_COMPONENT	Specifies color components to display. Attribute value is a combination of enumPixdumpComponentEx constants.

Remarks:

See IG_GUI_pixdump_attribute_get for more details.

1.3.1.5.158 enumPixdumpMode

GUI pixel dump window output mode.

Values:

IG_GUI_PIXDUMP_DIGITS_HEX	If this flag is set, pixel values are displayed in hexadecimal format.
IG_GUI_PIXDUMP_DATA_COLOR	Controls the display of the pixel dump for Indexed images. If this flag is set, palette values are displayed. Otherwise, raw pixel values are displayed.

1.3.1.5.159 enumPixel

Specifies data format for pixel access functions.

Values:

IG_PIXEL_PACKED	Values of several pixels can be packed in one byte.
IG_PIXEL_UNPACKED	Each pixel occupies at least one byte.
IG_PIXEL_RLE	Reserved for future use.

1.3.1.5.160 enumPixelate

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_PIXELATE_CENTER	This value has been deprecated and will be removed from the public API in a future release.
IG_PIXELATE_AVERAGE	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.161 enumPNGCompLevel

Specifies PNG compression level.

Values:

IG_PNG_MIN_COMPRESSION	Minimum level of PNG compression.
IG_PNG_MAX_COMPRESSION	Maximum level of PNG compression.
IG_PNG_DEFAULT_COMPRESSION	Default PNG compression level.

1.3.1.5.162 enumPNGStrip

Specifies PNG strip configurations.

Values:

IG_PNG_STRIP_FIXED_COUNT	Number of strips is fixed and every strip consists of equal number of rasters.
IG_PNG_STRIP_FIXED_BUFFER	Size of strip buffer is fixed. Number of rasters in each strip may vary.

1.3.1.5.163 enumPostScriptLevel

Specifies PostScrip format specifications known as Level 1, 2 or 3.

Values:

IG_PS_LEVEL_1	Support of PostScript level 1.
IG_PS_LEVEL_2	Support of PostScript level 2.
IG_PS_LEVEL_3	Support of PostScript level 3.

1.3.1.5.164 enumPostScriptType

Specifies the type of the output PostScript document. Used with TYPE control parameter of the POSTSCRIPT format filter.

Values:

IG_POSTSCRIPT	PostScript PS file format.
IG_EPS_NO_PREVIEW	PostScript EPS file format with no preview.
IG_EPS_STANDARD_PREVIEW	PostScript EPS file format with standard preview.
IG_EPS_EXTENDED_PREVIEW	PostScript EPS file format with extended preview.

1.3.1.5.165 enumPrintConstants

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_DSPL_PRINT_FULL_PAGE	This value has been deprecated and will be removed from the public API in a future release.
IG_DSPL_PRINT_THREE_QUARTER_PAGE	This value has been deprecated and will be removed from the public API in a future release.
IG_DSPL_PRINT_HALF_PAGE	This value has been deprecated and will be removed from the public API in a future release.
IG_DSPL_PRINT_QUARTER_PAGE	This value has been deprecated and will be removed from the public API in a future release.
IG_DSPL_PRINT_EIGHTH_PAGE	This value has been deprecated and will be removed from the public API in a future release.
IG_DSPL_PRINT_SIXTEENTH_PAGE	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.166 enumRampDirection

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_RAMP_FORWARD	This value has been deprecated and will be removed from the public API in a future release.
IG_RAMP_REVERSE	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.167 enumRampType

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_RAMP_HORIZONTAL	This value has been deprecated and will be removed from the public API in a future release.
IG_RAMP_VERTICAL	This value has been deprecated and will be removed from the public API in a future release.
IG_RAMP_PYRAMID	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.168 enumRasterPostProc

Specifies operation applied to each raster on image loading.

Values:

POST_PROCESS_ABIC_GREY_LUT	Apply ABIC gray look-up table to rasters.
POST_PROCESS_INVERT_BITONAL_RASTER	This value has been deprecated and will be removed from the public API in a future release.

1.3.1.5.169 enumRegionIS

Specifies the type of region stored in the clipboard.

Values:

IG_REGION_IS_RECT	The region available in clipboard is rectangle.
IG_REGION_IS_NON_RECT	The region available in clipboard is a non-rectangular area.
IG_REGION_IS_NOT_AVAIL	There is no image data in clipboard.

1.3.1.5.170 enumROI_IS

Specifies the types of non-rectangular Region of Interest (ROI).

Values:

IG_ROI_IS_RECTANGLE	The ROI is a rectangle.
IG_ROI_IS_ELLIPSE	The ROI is an ellipse.
IG_ROI_IS_POLYGON	The ROI is a polygon.

Remarks:

These modes are used with IG_IP_NR_ROI_to_HIGEAR_mask and describe what kind of non-rectangular ROI is passed in.

1.3.1.5.171 enumScrollTypes

Specifies scrollbars and scroll commands.

Values:

IG_DSPL_SCROLL_HORIZONTAL	Identifies horizontal scrollbar in the scrolling API.
IG_DSPL_SCROLL_VERTICAL	Identifies vertical scrollbar in the scrolling API.
IG_DSPL_HSCROLLBAR	Specifies a bitmask used to extract horizontal scrollbar attributes from scrolling mode value.
IG_DSPL_HSCROLLBAR_AUTO	Specifies that horizontal scrollbar is displayed automatically when needed.
IG_DSPL_HSCROLLBAR_ENABLE	Specifies that horizontal scrollbar is always displayed.
IG_DSPL_HSCROLLBAR_DISABLE	Specifies that horizontal scrollbar is always disabled and hidden.
IG_DSPL_VSCROLLBAR	Specifies a bitmask used to extract vertical scrollbar attributes from scrolling mode value.
IG_DSPL_VSCROLLBAR_AUTO	Specifies that vertical scrollbar is displayed automatically when needed.
IG_DSPL_VSCROLLBAR_ENABLE	Specifies that vertical scrollbar is always displayed.
IG_DSPL_VSCROLLBAR_DISABLE	Specifies that vertical scrollbar is always disabled and hidden.

1.3.1.5.172 enumShear

Specifies shear modes.

Values:

IG_SHEAR_HORIZONTAL	Shear horizontally.
IG_SHEAR_VERTICAL	Shear vertically.

1.3.1.5.173 enumTagTypes

Specifies data types for use with metadata tag callbacks.

Values:

IG_TAG_TYPE_NULL	No data - end of tags.
IG_TAG_TYPE_BYTE	Data is a 8 bit unsigned integer.
IG_TAG_TYPE_ASCII	Data is a 8 bit, NULL-terminated string.
IG_TAG_TYPE_SHORT	Data is a 16 bit unsigned integer.
IG_TAG_TYPE_LONG	Data is a 32 bit unsigned integer.
IG_TAG_TYPE_RATIONAL	Data is a pair of 32-bit unsigned integers, representing an unsigned rational number.
IG_TAG_TYPE_SBYTE	Data is a 8 bit signed integer.
IG_TAG_TYPE_UNDEFINED	Data is a 8 bit byte.
IG_TAG_TYPE_SSHORT	Data is a 16-bit signed integer.
IG_TAG_TYPE_SLONG	Data is a 32-bit signed integer.
IG_TAG_TYPE_SRATIONAL	Data is a pair of 32-bit signed integers, representing a signed rational number.
IG_TAG_TYPE_FLOAT	Data is a 4-byte single-precision IEEE floating point number.
IG_TAG_TYPE_DOUBLE	Data is a 8-byte double-precision IEEE floating point number.
IG_TAG_TYPE_RAWBYTES	Data is a series of raw data bytes.
IG_TAG_TYPE_LONGARRAY	Data is an array of 32-bit signed integers.
IG_TAG_TYPE_UNICODE	Data is a UNICODE string, 16 bit WCHARs terminated by two NULLs.
IG_TAG_TYPE_FILETIME	Data is a 64 bit FILETIME structure.
IG_TAG_TYPE_DATE	Data is a 64 bit DATE structure.

1.3.1.5.174 enumThreadLockMode

Specifies thread access lock modes, used by IG_thread_image_lock and IG_thread_image_unlock functions.

Values:

IG_THREAD_LOCK_READ	Thread requested a read lock for the image.
IG_THREAD_LOCK_WRITE	Thread requested a write lock for the image.

1.3.1.5.175 enumTIFFBitonalPaletteMode

This enumeration specifies whether ImageGear shall fix strange looking palettes when reading bi-tonal TIFF images.

Values:

IG_TIF_BITONAL_PALETTE_MODE_LEGACY	Keep ImageGear 16.0 behavior: read all 1-bit palettes as is; if palette is missing, assume increasing (blackzero) palette.
IG_TIF_BITONAL_PALETTE_MODE_KEEP_AS_IS	Keep palette as is, even if it is all-black or red-green. If photometric interpretation is PALETTE_COLOR, but COLORMAP tag is absent, assume all-black palette.
IG_TIF_BITONAL_PALETTE_MODE_FIX	Fix strange looking palettes, as follows: if (R0+G0+B0)/3 < (R1+G1+B1)/3 change palette to (black, white). Otherwise, change palette to (white, black). Specifically, constant palettes (all-black, all-white) are replaced with (white, black) palette.

1.3.1.5.176 enumTIFFPhoto

Specifies TIFF photometric interpretations. Used with TIFF PHOTOMETRIC control parameter.

Values:

IG_TIF_PHOTO_WHITEZERO	Indicates that zero 0 pixel value represents white color.
IG_TIF_PHOTO_BLACKZERO	Indicates that zero 0 pixel value represents black color.
IG_TIF_PHOTO_RGB	Indicates RGB colorspace.
IG_TIF_PHOTO_PALETTE	Indicates indexed colors image with RGB palette.
IG_TIF_PHOTO_TRANSPARENCY	Indicates transparency mask.
IG_TIF_PHOTO_CMYK	Indicates CMYK colorspace.
IG_TIF_PHOTO_YCBCR	Indicates YCBCR colorspace.
IG_TIF_PHOTO_CIELAB	Indicates CIELAB colorspace.

Remarks:

See TIFF 6 specification for more information.

1.3.1.5.177 enumTIFFWriteConfig

Specifies values for saving stripped and tiled TIFF images. Used with TIFF WRITE_CONFIG control parameter.

Values:

IG_TIF_STRIP_FIXED_COUNT	Write the image using a fixed number of strips. The number of strips to use can be set via the NUMBER_OF_STRIPS control parameter.
IG_TIF_STRIP_FIXED_BUFFER	Write the image using strips so that each strip is not greater than the specified size in bytes. The size of the strip buffer can be set via the BUFFER_SIZE control parameter. Note that at least one raster will be included in the strip.
IG_TIF_TILED_FIXED_SIZE	Save the image using tiles of fixed size. The size of the tiles can be set via the TILE_WIDTH and TILE_HEIGHT control parameters.
IG_TIF_TILED_FIXED_COUNT	Save the image using a fixed number of tiles. The number of tiles in both the horizontal and vertical direction can be set via the control parameters TILE_H_COUNT and TILE_V_COUNT.

1.3.1.5.178 enumXWDType

This enumeration has been deprecated and will be removed from the public API in a future release.

Values:

IG_XWD_TYPE_XYBITMAP	This value has been deprecated and will be removed from the public API in a future release.
IG_XWD_TYPE_XYPIXMAP	This value has been deprecated and will be removed from the public API in a future release.
IG_XWD_TYPE_ZPIXMAP	This value has been deprecated and will be removed from the public API in a future release.

1.3.2 MD Component API Reference

This section provides information about the ImageGear Medical component.

You can call MD component functions in two ways.

First, you may call this function through it macro defined in i_MED.h public header file:

```
#define MED_DCM_load_DICOM(_lpFileName, _lphIGear, _nSyntax, _page_number)\ (AT_ERRCOUNT
(CACCUAPI *)(const LPSTR, \const LPCHAR, LPHIGEAR, const AT_MODE, const UINT)) \
          IG_comm_function_call)("MED.MED_DCM_load_DICOM", \
(_lpFileName),(_lphIGear),(_nSyntax), (_page_number))
```

In this case your application will search all included public headers for this macro and call this function through IG_comm_function_call() component manager function call that is also determined in i_MED.h.

So, if you are going to use MD component function in a multiple loop, we recommend the second way of using this function type declaration that is also determined in i_MED.h file:

```
typedef  AT_ERRCOUNT (LPACCUAPI LPAFT_MED_DCM_LOAD_DICOM)(
const LPSTRlpsz                 FileName,
LPHIGEAR                        lphIGear,
const AT_MODE                   nSyntax,
const UINT                      page_number
 );
```

So you should declare the variable of this function type, and then use IG_comm_entry_request() function to initialize this variable with the correct value, and call it then.

This section provides information about the following:

- MD Component Functions Reference
- MD Component Macros Reference
- MD Component Structures Reference
- MD Component Enumerations Reference

1.3.2.1 MD Component Functions Reference

This section provides information about the MD Component Functions, arranged in alphabetical order within functional groups.

- Data Set Functions
- Display Functions
- File Functions
- Image Processing Functions
- Modality Transform Functions
- Overlay Functions
- Presentation State Functions
- Utility Functions

1.3.2.1.1 Data Set Functions

This section provides information about the Data Set group of functions.

- MED_DCM_DS_bits_get
- MED_DCM_DS_copy_get
- MED_DCM_DS_create
- MED_DCM_DS_curr_data_get
- MED_DCM_DS_curr_data_get_string
- MED_DCM_DS_curr_data_set
- MED_DCM_DS_curr_index_get
- MED_DCM_DS_curr_info_get
- MED_DCM_DS_curr_remove
- MED_DCM_DS_DE_insert
- MED_DCM_DS_destroy
- MED_DCM_DS_exists
- MED_DCM_DS_info_get
- MED_DCM_DS_is_empty
- MED_DCM_DS_LUT_copy_get
- MED_DCM_DS_LUT_exists
- MED_DCM_DS_LUT_update_from
- MED_DCM_DS_move_ascend
- MED_DCM_DS_move_descend
- MED_DCM_DS_move_find
- MED_DCM_DS_move_find_first
- MED_DCM_DS_move_first
- MED_DCM_DS_move_index
- MED_DCM_DS_move_last
- MED_DCM_DS_move_next
- MED_DCM_DS_move_prev
- MED_DCM_DS_orig_TS_get
- MED_DCM_DS_part10_get
- MED_DCM_DS_part10_set
- MED_DCM_DS_PixPadVal_get
- MED_DCM_DS_PixPadVal_set
- MED_DCM_DS_preamble_get
- MED_DCM_DS_preamble_set
- MED_DCM_DS_Rescale_get
- MED_DCM_DS_TS_get
- MED_DCM_DS_TS_set
- MED_DCM_DS_update_file
- MED_DCM_DS_update_from
- MED_DCM_DS_Window_Level_get
- MED_DCM_DS_Window_Level_get_64

1.3.2.1.1.1 MED_DCM_DS_bits_get

This function returns three critical DE values from the Data Set: the number of Bits Allocated (0028,0100), the number of Bits Stored (0028,0101), and the position of the High Bit (0028,0102). These values are stored in the HDS structure pointed to by the HIGEAR.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_bits_get(
        const HIGEAR hIGear,
        LPUINT lpBitsAllocated,
        LPUINT lpBitsStored,
        LPUINT lpHighBit,
        LPUINT lpSamplesPerPix);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle of the image containing a DICOM Data Set.
lpBitsAllocated	LPUINT	A far pointer that returns the number of Bits Allocated (0028,0100) for each pixel in the image. Set this to NULL if you do not need this information.
lpBitsStored	LPUINT	A far pointer that returns the number of Bits Stored (0028,0101) for each pixel in the image. Set this to NULL if you do not need this information.
lpHighBit	LPUINT	A far pointer that returns the position of the High Bit (0028,0102) of the pixels in a DICOM image. Set this to NULL if you do not need this information.
lpSamplePerPix	LPUINT	A far pointer that returns the number of Samples Per Pixel (0028,0002) for the DICOM image. Set this to NULL if you do not need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

The value returned by lpBitsAllocated is not always the same as the number of bits per pixel for the image's DIB. It is the number of bits allocated per sample for each pixel. A 24-bit RGB image would return a Bits Allocated of 8 since each of the 3 samples has 8 bits allocated for it.

The Bits Stored is the number of bits actually used out of the total available (Bits Allocated). You can have a 16-bit grayscale image that only actually uses 12-bits. In this case, the Bits Allocated would be 16 and Bits Stored would be 12. The Bits Stored is always less than or equal to the Bits Allocated.

The High Bit shows where the Bits Stored are placed in the Bits Allocated WORD or DWORD. Since the Bits Stored can be less than the Bits Allocated, the Bits Stored can be placed in the Bits Allocated with different starting points. This value tells you where the Bits Stored actually resides. High Bits is always less than Bits Allocated.

See Also

MED_IP_high_bit_transform

MED_IP_reduce_depth_with_LUT

MED_IP_reduce_depth_with_downshift

1.3.2.1.1.2 MED_DCM_DS_copy_get

This function allocates a new Element List and copies contents of the Data Set associated with HIGEAR to it.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_copy_get(
       HIGEAR hIGear,
       HIGMEDELEMLIST* lphDstList);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to an image from which the Element List will be copied.
lphDstList	HIGMEDELEMLIST*	Address of the HIGMEDELEMLIST handle to the Data Element List object.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

1.3.2.1.1.3 MED_DCM_DS_create

This function creates a Data Set.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_create(
        HIGEAR hIGear,
        const AT_MODE Transfer_syntax);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
Transfer_syntax	const AT_MODE	Set this variable to the desired Transfer Syntax (TS) with which to create the Data Set. Use one of the ImageGear defined constants defined in enumIGMedTS, such as: MED_DCM_TS_IMPLICIT_VR_LE, MED_DCM_TS_EXPLICIT_VR_LE, MED_DCM_TS_EXPLICIT_VR_BE.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
HIGEAR                              hIGear;
AT_MODE                                     Transfer_syntax;
IG_load_file("image1.tif", &hIGear);
Transfer_syntax = MED_DCM_TS_IMPLICIT_VR_LE;
MED_DCM_DS_create(hIGear, Transfer_syntax);
```

Remarks:

If there is already a Data Set that is associated with the HIGEAR image, it will be replaced. In addition, the absolute minimum of Critical DEs (Data Elements), such as Pixel Representation and Samples per Pixel will be automatically added to the Data Set.

Below is a list of those Mandatory DEs that will be filled out automatically by this function. See Part 6:Data Dictionary of the DICOM Specification for the definitions of these DEs.

Note that there are two different Group Numbers listed below, and that the Tag numbers (the second numbers shown in the parentheses) identify which Data Element will be filled in:

```
        Affected DEs from Tag.group                              = 0x0028:
        (0028,0010)                     /* Rows          */
        (0028,0011)                     /* Columns               */
        (0028,0100)                     /* Bits Allocated                        */
        (0028,0101)                     /* Bits Stored                   */
        (0028,0102)                     /* High Bit                      */
        (0028,0004)                     /* Photometric Interpretation
*/
        (0028,0103)                     /* Pixel Representation                          */
        (0028,0002)                    /* Samples per pixel              */
        (0028,0006)                    /* Planar Configuration                          */
        Affected DEs from Tag.group                              = 0x7FE0;
        (7FE0,0010)                    /* Pixel Data                     */
```

See Working With DICOM Data Structures section for more information.

1.3.2.1.1.4 MED_DCM_DS_curr_data_get

This function returns the data (Value Field) of the Current Data Element in its native form.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_data_get(
        const HIGEAR hIGear,
        LPVOID lpData,
        const DWORD size_of_lpData,
        LPAT_DCM_VL lpActualSize);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image from which to get data.
lpData	LPVOID	A far pointer to a VOID buffer into which the data will be copied.
size_of_lpData	const DWORD	Size of above buffer, lpData.
lpActualSize	LPAT_DCM_VL	Actual number of bytes copied to lpData (NULL if inconsequential). This will always be equal to or smaller than size_of_lpData. If lpActualSize is less than the VL of the current Data Element then lpData does not contain all the data because lpData does not point to enough memory to hold the entire object.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

Each Data Element can be of one of many different data types (int, word, bytes, float, double, string, etc.) You must know the VR (Value Representation) of the data in order to use this data. To get the VR and the VL (Value Length), you can use one of the MED_DCM_DS_move_...() functions or MED_DCM_DS_curr_info_get(). Each time you move the Current Data Element with a _move_...() function, the VR and VL of the new Current Data Element are returned to you.

To retrieve the data as a string use MED_DCM_DS_curr_data_get_string().

Some VRs depend on the byte order of your operating system (Big Endian or Little Endian) and the Transfer Syntax of the DICOM file. Examples of such VRs are: WORD, LONG, FLOAT, DOUBLE, etc. The Data Field values will be returned to you already adjusted to the proper format and no byte-swapping is needed.

1.3.2.1.1.5 MED_DCM_DS_curr_data_get_string

This function gets the data from the Current Data Element, and always returns it to you as a NULL-terminated character string.

Declaration:

```
BOOL ACCUAPI MED_DCM_DS_curr_data_get_string (
       const HIGEAR hIGear,
       LPCHAR lpString,
       const DWORD size_of_lpstring
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image from which to get data.
lpString	LPCHAR	A far pointer to a memory location that will be filled with the data from the Current Data Element as a NULL-terminated character string.
size_of_lpString	const DWORD	The variable which tells the function the length of lpString, in bytes, expressed as a DWORD. If you specify a string length that is not long enough to hold the data, the data will simply be truncated.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

The marker to the Current Data Element can be moved using one of the MED_DCM_DS_move_...() functions.

Data Elements can be of many different data types (int, word, bytes, float, double, string, etc.), but this function will always convert the data to a string. To return the data in its "natural form", use MED_DCM_DS_curr_data_get().

1.3.2.1.1.6 MED_DCM_DS_curr_data_set

This function allows you to overwrite the Value Field (data) of the Current Data Element by copying the data from your buffer to the HDS table.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_data_set(
        HIGEAR hIGear,
        const LPVOID lpData,
        const DWORD size_of_data
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image.
lpData	const LPVOID	A far pointer of type VOID. Set this to the data you would like stored into the Data Field of the Current Data Element.
size_of_data	const DWORD	Set this DWORD variable to the size of the data in lpData.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

The Data must be of the correct data type to match the Value Representation (VR) of the Current DE. You can use MED_DCM_DS_curr_info_get()to find out the VR of the Current DE, and you can use one of the MED_DCM_DS_move_...() functions to set the Current Data Element.

If the Data Value can accept multiple Data Elements, the data values should be set as a single block of memory. To query the Value Multiplicity (VM) use MED_DCM_util_tag_info_get().

The length of a DICOM Data Field must always be an even number. If you set the size_of_lpData to an odd number of bytes, ImageGear will pad it (and your data) to make it an even-numbered length.

An error is set if the data type does not match the VR of the Current DE.

1.3.2.1.1.7 MED_DCM_DS_curr_index_get

Returns the index of the Current Data Element in the Data Set associated with the current HIGEAR image.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_index_get(
        const HIGEAR hIGear,
        LPDWORD lpIndex
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	The HIGEAR handle to the image being queried.
lpIndex	LPDWORD	A far pointer of type DWORD that returns with the index of the Current Data Element.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This function indexes through all levels of the Data Set. The index of the first Data Element in the table is 0.

To move to a specific index and make it the Current Data Element, use MED_DCM_DS_move_index().

1.3.2.1.1.8 MED_DCM_DS_curr_info_get

This function returns information about the Current Data Element (DE).

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_info_get(
       const HIGEAR hIGear,
       LPAT_DCM_TAG lpTag,
       LPAT_DCM_VR lpVR,
       LPAT_DCM_VL lpVL,
       LPWORD lpLevel,
       LPDWORD lpItem_count
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
lpTag	LPAT_DCM_TAG	A far pointer that returns a 32-bit value of type AT_DCM_TAG indicating the numerical value of the Tag of the Current Data Element; set this to NULL if you do not need this information. The numerical values of the DICOM Tags are defined in the enumIGMedTag enumeration.
lpVR	LPAT_DCM_VR	A far pointer to variable of type AT_DCM_VR which returns the Value Representation (VR) of the Current Data Element; set this to NULL if you do not need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer to a variable of type DWORD that returns the length, in bytes, of the Current Data Element's Data Field; set this to NULL if you do not need this information.
lpLevel	LPWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL if you do not need this information.
lpItem_count	LPDWORD	A far pointer that returns the number of items stored in the Data Field; set this to NULL if you do not need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This function operates in just the same way as the MED_DCM_move_...() functions, except that it does not change the Current DE - it only reports about it.

1.3.2.1.1.9 MED_DCM_DS_curr_remove

This function removes the Current Data Element from the Data Set.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_curr_remove (
       HIGEAR hIGear,
       LPBOOL lpRemoved
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
lpRemoved	LPBOOL	A far pointer to a BOOL, which returns TRUE if the Current Data Element was removed; and will return FALSE if there are no more removable DEs in the Data Set or if the Current Data Element could not be removed.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

The new Current Data Element will be the Data Element following the one that was just removed, unless you have deleted the last DE, in which case the CDE will be the "new" last DE.

If the Data Set is empty (there are no removable DEs remaining) lpRemoved will return FALSE. The critical DEs cannot be removed. Non-removable DEs also include Sequence and Item Delimiters.

1.3.2.1.1.10 MED_DCM_DS_DE_insert

This function inserts a Data Element into the Data Set.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_DE_insert(
        HIGEAR hIGear,
        const AT_DCM_TAG Tag,
        const AT_DCM_VR vr,
        const LPVOID lpData,
        const DWORD size_of_data
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
Tag	const AT_DCM_TAG	Set to a Tag value. The Tag must be supplied as a 32-bit value in which the first 16 bits (WORD) represent the Group Number and the second 16 bits represent the Element Number. Public DICOM tags are listed in enumIGMedTag enumeration.
vr	const AT_DCM_VR	Set to the VR (Value Representation) of the Data Element to be inserted. See enumIGMedVR for possible VR values.
lpData	const LPVOID	A far VOID pointer to the data that you would like to insert.
size_of_data	const DWORD	Set this DWORD variable to the size of the data in lpData.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
HIGEAR          hIGear;
MED_DCM_DS_DE_insert(hIGear, DCM_TAG_PhotometricInterpretation, MED_DCM_VR_CS,
"MONOCHROME2", 11);
```

Remarks:

Your new Data Element will be placed into the Data Set sorted by its Tag value on the same level as that of the Current Data Element. If the DE already exists, the new one overwrites it. Specifying a Group Length DE does not cause an error, but will simply be ignored. Your data will be padded to an even length if necessary.

Currently, there are no constants defined for those Data Elements that have a VR of "CS." Refer to Part 3 of the Specification for the valid Code Strings which you can enter for data of type CS (Code String). Note also that the length of a Code String is the number of characters between the parentheses.

1.3.2.1.1.11 MED_DCM_DS_destroy

This function destroys the Data Set associated with the HIGEAR image specified.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_destroy(HIGEAR hIGear);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image whose Data Set will be destroyed.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

Once the Data Set is removed, the image is just like any other image loaded into ImageGear.

1.3.2.1.1.12 MED_DCM_DS_exists

This function determines whether an image has a Data Set associated with it.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_exists(
        const HIGEAR hIGear,
        LPBOOL lpExists
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	A HIGEAR handle to an image.
lpExists	LPBOOL	A far pointer that returns a BOOL. If it returns TRUE, a Data Set exists for the image; if it returns FALSE, a Data Set does not exist for the image.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Supply it with the HIGEAR handle to the image you want to check, and lpExists will return whether or not this image has a Data Set. See Working With DICOM Data Structures section for more information.

1.3.2.1.1.13 MED_DCM_DS_info_get

This function returns the number of Data Elements (DEs) associated with the Data Set of the HIGEAR image.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_info_get(
        const HIGEAR hIGear,
        LPDWORD lpNumTags,
        LPDWORD lpMaxLevel
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to an image.
lpNumTags	LPDWORD	A far pointer to a DWORD which returns the number of Data Elements (same as the number of Tags) associated with the image's Data Set.
lpMaxLevel	LPDWORD	A far pointer to a DWORD which returns the maximum SQ Level of the Data Set.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

You might use the number of DEs returned to set the limit on a loop that iterates through each Data Element in a Data Set. If the DataSet contains SQ (Sequence of Items) data elements, the function returns the total number of data elements, including data elements contained within sequences. If there are no SQ Data Elements, lpMaxLevel is set to 0.

1.3.2.1.1.14 MED_DCM_DS_is_empty

This function returns a TRUE through lpIsEmpty argument if the Data Set associated with HIGEAR has no Data Elements in it.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_is_empty(
        const HIGEAR hIGear,
        LPBOOL lpIsEmpty
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
lpIsEmpty	LPBOOL	A far pointer to a BOOL which returns the status of the Tags in the Data Set.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

1.3.2.1.1.15 MED_DCM_DS_LUT_copy_get

This function obtains a new copy of a specified LUT from either presentation state HIGEAR (hIGearPresState), or the image HIGEAR (hIGear).

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_LUT_copy_get(
        HIGEAR hIGear,
        HIGEAR hIGearPresstate,
        AT_DCM_TAG lutSqTag,
        HIGLUT* lpLUT
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image whose dataset is checked for presence of the LUT.
hIGearPresstate	HIGEAR	Presentation state HIGEAR whose dataset is checked for presence of the LUT. Set to NULL if no presentation state HIGEAR is available.
lutSqTag	AT_DCM_TAG	Specifies the LUT sequence.
lpLUT	HIGLUT*	Returns new HIGLUT object with the LUT obtained from a DataSet.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

If the LUT exists in both the Presentation state HIGEAR, and in the image HIGEAR, the function returns the LUT from Presentation State HIGEAR.

Use IG_LUT_destroy() to destroy the LUT returned from this function when it is no longer needed.

This function supports the following LUT sequences:

- DCM_TAG_ModalityLUTSequence
- DCM_TAG_VOILUTSequence
- DCM_TAG_PresentationLUTSequence

1.3.2.1.1.16 MED_DCM_DS_LUT_exists

This function checks whether a presentation state HIGEAR (hIGearPresState), or the image HIGEAR (hIGear) contain specified LUT sequence.

Declaration:

```
AT_BOOL ACCUAPI MED_DCM_DS_LUT_exists(
        HIGEAR hIGear,
        HIGEAR hIGearPresstate,
        AT_DCM_TAG lutSqTag
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image whose dataset is checked for presence of the LUT.
hIGearPresstate	HIGEAR	Presentation state HIGEAR whose dataset is checked for presence of the LUT. Set to NULL if no presentation state HIGEAR is available.
lutSqTag	AT_DCM_TAG	Specifies the LUT sequence.

Return Value:

TRUE if the DataSet attached to the HIGEAR has a specified LUT sequence; FALSE otherwise.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This function supports the following LUT sequences:

- DCM_TAG_ModalityLUTSequence
- DCM_TAG_VOILUTSequence
- DCM_TAG_PresentationLUTSequence

1.3.2.1.1.17 MED_DCM_DS_LUT_update_from

This function adds specified LUT to the DataSet.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_LUT_update_from(
        HIGEAR hIGear,
        AT_DCM_TAG lutSqTag,
        HIGLUT lut
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
lutSqTag	AT_DCM_TAG	LUT sequence.
lut	HIGLUT	LUT handle.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

The function supports the following LUTs:

LUT	Sequence Tag
Modality LUT	DCM_TAG_ModalityLUTSequence
VOI LUT	DCM_TAG_VOILUTSequence, DCM_TAG_SoftcopyVOILUTSequence
Presentation LUT	DCM_TAG_PresentationLUTSequence

1.3.2.1.1.18 MED_DCM_DS_move_ascend

This function moves the Current Data Element up one level.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_ascend(
        HIGEAR hIGear,
        LPAT_DCM_TAG lpTag,
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VL lpVL,
        LPWORD lpLevel,
        LPDWORD lpICount
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
lpTag	LPAT_DCM_TAG	A far pointer that returns a 32-bit value of type AT_DCM_TAG indicating the numerical value of the Tag of the Current Data Element; set this to NULL if you do not need this information. The numerical values of the DICOM Tags are defined in enumIGMedTag enumeration.
lpVR	LPAT_DCM_VR	A far pointer which returns the new current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer which returns the length of the Data Field, in bytes.
lpLevel	LPWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL, if you do not need this information.
lpICount	LPDWORD	Returns the Item Count of the data; set to NULL if you don't need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This is only applicable if the Current Data Element is positioned within an SQ Data Element or within an Item Data Element.

If the Current Data Element is positioned anywhere in an SQ, the Current Data Element ascends to the SQ. If within an Item, the new Current DE becomes the Item DE. Only one level is ascended per call.

If the Current DE is at the top level (0), no action is taken.

If this function is successful, the lpLevel decreases in value by 1. Zero refers to the top level. As the number gets larger, the Data Element is deeper into the hierarchy. Data Elements (as well as SQs and Item Delimiters) are always stored in even-numbered levels; odd-numbered levels contain Items and SQ Delimiters.

1.3.2.1.1.19 MED_DCM_DS_move_descend

This function moves the Current Data Element down one level.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_descend (
        const HIGEAR hIGear,
        LPAT_DCM_TAG lpTag,
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VL lpVL,
        LPWORD lpLevel,
        LPDWORD lpICount);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
lpTag	LPAT_DCM_TAG	A far pointer that returns a 32-bit value of type AT_DCM_TAG indicating the numerical value of the Tag of the Current Data Element; set this to NULL if you do not need this information. The numerical values of the DICOM Tags are defined in enumIGMedTag enumeration.
lpVR	LPAT_DCM_VR	A far pointer which returns the new current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer which returns the length of the Data Field, in bytes.
lpLevel	LPWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL, if you do not need this information.
lpICount	LPDWORD	Returns the Item Count of the data; set to NULL if you don't need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This is only applicable if the Current Data Element is positioned at an SQ Data Element or at an Item Data Element.

If the Current Data Element is positioned at an SQ Data Element, the Current Data Element will descend to the first Item in the sequence. If at an Item, the new Current DE becomes the first DE within the Item. Only one level is descended per call.

If the Current DE does not point to an SQ or Item DE, no action is taken and lpLevel returns the Current Level.

If this function is successful, the lpLevel increments from its Current DE level. Zero refers to the top level. As the number gets larger the Data Element is deeper into the hierarchy. Data Elements (as well as SQs and Item Delimiters) are always stored in odd-numbered levels; even-numbered levels contain Items and SQ Delimiters.

1.3.2.1.1.20 MED_DCM_DS_move_find

This function searches the Data Set associated with hIGear for the Tag specified in lpTag.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_find(
        HIGEAR hIGear,
        const AT_MODE level_op,
        const AT_DCM_TAG Tag,
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VL lpVL,
        LPDWORD lpLevel,
        LPDWORD lpICount,
        LPBOOL lpTagFound
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
level_op	const AT_MODE	A variable of type AT_MODE that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: • MED_DCM_MOVE_LEVEL_FIXED: This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ, you can only move about the SQ. • MED_DCM_MOVE_LEVEL_FLOAT: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ, the Current DE moves down into it. At the end of the SQ the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
Tag	const AT_DCM_TAG	Set to a value of type DWORD that identifies the Tag value for which you would like to search. The first 16 bits of the DWORD represent the Group Number; the second 16 bits represent the Element Number.
lpVR	LPAT_DCM_VR	A far pointer which returns the new current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer which returns the length of the Data Field, in bytes.
lpLevel	LPDWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL if you do not need this information.
lpICount	LPDWORD	Returns the Item Count of the data; set to NULL if you don't need this information.
lpTagFound	LPBOOL	Returns TRUE is the Tag was found; FALSE otherwise.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

If the Tag is found, the Data Element becomes the Current Data Element. This function also returns the VR (Value Representation), the number of bytes in the Tag's data, and the Item Count.

The levels of the Data Set that will be searched depends on the setting of level_op.

1.3.2.1.1.21 MED_DCM_DS_move_find_first

This function searches the Data Set associated with the HIGEAR image for the first Tag of the Group Number specified by GroupNum.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_find_first(
        const HIGEAR hIGear,
        const AT_MODE level_op,
        const WORD GroupNum,
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VL lpVL,
        LPDWORD lpLevel,
        LPDWORD lpItem_count,
        LPBOOL lpTagFound
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle of the image.
level_op	const AT_MODE	A variable of type AT_MODE that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: • MED_DCM_MOVE_LEVEL_FIXED: This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ, you can only move about the SQ. • MED_DCM_MOVE_LEVEL_FLOAT: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ, the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
GroupNum	const WORD	Set this WORD variable to the Group Number for which to search.
lpVR	LPAT_DCM_VR	A far pointer which returns the new current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer which returns the length of the Data Field, in bytes.
lpLevel	LPDWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL, if you do not need this information.
lpICount	LPDWORD	Returns the Item Count of the data; set to NULL if you don't need this information.
lpTagFound	LPBOOL	Returns TRUE is the Tag was found; FALSE otherwise.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

When the first Tag with group GroupNum is found, the Data Element is made the Current Data Element. This function also returns the VR (Value Representation), the number of bytes in the Tag's data, and the Item Count.

Which levels of the Data Set will be considered depends on the setting of level_op.

1.3.2.1.1.22 MED_DCM_DS_move_first

This function makes the first Data Element in the specified level the Current Data Element.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_first(
        const HIGEAR hIGear,
        const AT_MODE level_op,
        LPAT_DCM_TAG lpTag,
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VL lpVL,
        LPWORD lpLevel,
        LPDWORD lpICount
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
level_op	const AT_MODE	A variable of type AT_MODE that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: • MED_DCM_MOVE_LEVEL_FIXED: This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ, you can only move about the SQ. • MED_DCM_MOVE_LEVEL_FLOAT: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ, the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
lpTag	LPAT_DCM_TAG	A far pointer that returns a 32-bit value of type AT_DCM_TAG indicating the numerical value of the Tag of the Current Data Element; set this to NULL if you do not need this information. The numerical values of the DICOM Tags are defined in enumIGMedTag enumeration.
lpVR	LPAT_DCM_VR	A far pointer which returns the new current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer which returns the length of the Data Field, in bytes.
lpLevel	LPWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL, if you do not need this information.
lpICount	LPDWORD	Returns the Item Count of the data; set to NULL if you don't need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This function also returns the DE's VR (Value Representation), the number of bytes in the Tag's data, and the Item Count.

Which level of the Data Set will be considered depends upon the setting of level_op.

1.3.2.1.1.23 MED_DCM_DS_move_index

This function moves the Current Data Element to the Data Element in the Data Set indicated by index, which represents a particular index in the array of Data Elements that make up the Data Set.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_index(
        const HIGEAR hIGear,
        const DWORD index,
        LPAT_DCM_TAG lpTag,
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VL lpVL,
        LPDWORD lpLevel,
        LPDWORD lpICount
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	The HIGEAR handle to the image.
index	const DWORD	Set to a DWORD that indicates the index that you would like to become the new Current Data Element.
lpTag	LPAT_DCM_TAG	A far pointer that returns a 32-bit value of type AT_DCM_TAG indicating the numerical value of the Tag of the Current Data Element; set this to NULL if you do not need this information. The numerical values of the DICOM Tags are defined in enumIGMedTag enumeration.
lpVR	LPAT_DCM_VR	A far pointer which returns the new current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer which returns the length of the Data Field, in bytes.
lpLevel	LPDWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL, if you do not need this information.
lpICount	LPDWORD	Returns the Item Count of the data; set to NULL if you don't need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This function also returns the Tag's VR (Value Representation), its Tag value, the number of bytes in the Tag's data, and the Item Count. The index refers to which element in the Data Element array should be used. The first Data Element will always have an index of 0, and the last Data Element will always have an index of (Total # of DEs - 1). Use MED_DCM_DS_info_get() to find out the total # of Data Elements associated with an image's Data Set.

This function pays no attention to Levels. It only returns the level of the Current Data Element.

1.3.2.1.1.24 MED_DCM_DS_move_last

This function moves the Current Data Element to the last Data Element in the Data Set, or in the current level, depending on level_op setting.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_last(
        const HIGEAR hIGear,
        const AT_MODE level_op,
        LPAT_DCM_TAG lpTag,
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VL lpVL,
        LPWORD lpLevel,
        LPDWORD lpICount
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
level_op	const AT_MODE	A variable of type AT_MODE that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: • MED_DCM_MOVE_LEVEL_FIXED: This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ, you can only move about the SQ. • MED_DCM_MOVE_LEVEL_FLOAT: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ, the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
lpTag	LPAT_DCM_TAG	A far pointer that returns a 32-bit value of type AT_DCM_TAG indicating the numerical value of the Tag of the Current Data Element; set this to NULL if you do not need this information. The numerical values of the DICOM Tags are defined in enumIGMedTag enumeration.
lpVR	LPAT_DCM_VR	A far pointer which returns the new current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer which returns the length of the Data Field, in bytes.
lpLevel	LPWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL, if you do not need this information.
lpICount	LPDWORD	Returns the Item Count of the data; set to NULL if you don't need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

1.3.2.1.1.25 MED_DCM_DS_move_next

This function moves the Current Data Element to the next Data Element in the Data Set.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_next(
        const HIGEAR hIGear,
        const AT_MODE level_op,
        LPAT_DCM_TAG lpTag;
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VL lpVL,
        LPWORD lpLevel,
        LPDWORD lpICount,
        LPLONG lpNumRemaining
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
level_op	const AT_MODE	A variable of type AT_MODE that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: • MED_DCM_MOVE_LEVEL_FIXED: This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ, you can only move about the SQ. • MED_DCM_MOVE_LEVEL_FLOAT: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ, the Current DE will move back out to the lower levels (for example, from Level 2 to Level 1).
lpTag	LPAT_DCM_TAG	A far pointer that returns a 32-bit value of type AT_DCM_TAG indicating the numerical value of the Tag of the Current Data Element; set this to NULL if you do not need this information. The numerical values of the DICOM Tags are defined in enumIGMedTag enumeration.
lpVR	LPAT_DCM_VR	A far pointer which returns the new current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer which returns the length of the Data Field, in bytes.
lpLevel	LPWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL, if you do not need this information.
lpICount	LPDWORD	Returns the Item Count of the data; set to NULL if you don't need this information.
lpNumRemaining	LPLONG	A far pointer to a LONG that returns the number of DE remaining until the end is reached. If the returned value is 0, you are now at last DE, if the returned value is -1, you have attempted to move past the last DE.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR                  hIGear;
BOOL                    IsLast;
AT_DCM_TAG                       lpTag;
AT_DCM_VR                       lpVR;
AT_DCM_VL                       lpVL;
DWORD                   lpICount;
DWORD                   size_of_lpData;
IsLast = FALSE:
  MED_DCM_DS_move_first(hIGear, &lpTag, &lpVR, &lpVL, &lpICount);
/* iterate through all Data Elements returning the Tag, VR, VL and item count of each one.
End the loop when lpIsLast == TRUE */
while (IsLast == FALSE) {
        MED_DCM_DS_curr_data_get(hIGear, lpData, size_of_lpData);
        MED_DCM_DS_move_next(hIGear, &lpTag, &lpVR, &lpVL, &lpICount, &lpIsLast);
```

```
   }
```

Remarks:

The value of lpNumRemaining tells you whether the Tag has now become the last Tag in the list.

The level_op setting determines whether the Current Data Element can move from level to level, or it has to stay on the same level.

1.3.2.1.1.26 MED_DCM_DS_move_prev

This function moves the Current Data Element to the previous Data Element in the Data Set.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_move_prev(
        const HIGEAR hIGear,
        const AT_MODE level_op,
        LPAT_DCM_TAG lpTag;
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VL lpVL,
        LPWORD lpLevel,
        LPDWORD lpICount,
        LPLONG lpNumRemaining
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
level_op	const AT_MODE	A variable of type AT_MODE that tells the function how to move when it comes to an SQ. SQs are like indented outline items, allowing for hierarchies of Data Elements. Set this to one of the following constants: • MED_DCM_MOVE_LEVEL_FIXED: This setting tells the function to move only within the same level as the previous Current DE. An SQ and all its Data Elements are skipped over. If you are in a SQ, you can only move about the SQ. • MED_DCM_MOVE_LEVEL_FLOAT: This setting tells the function to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ, the Current DE moves back out to the lower levels (for example, from Level 2 to Level 1).
lpTag	LPAT_DCM_TAG	A far pointer that returns a 32-bit value of type AT_DCM_TAG indicating the numerical value of the Tag of the Current Data Element; set this to NULL if you do not need this information. The numerical values of the DICOM Tags are defined in enumIGMedTag enumeration.
lpVR	LPAT_DCM_VR	A far pointer which returns the new current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVL	LPAT_DCM_VL	A far pointer which returns the length of the Data Field, in bytes.
lpLevel	LPWORD	A far pointer to a WORD which returns the level in the hierarchy of the new Current Data Element; set to NULL, if you do not need this information.
lpICount	LPDWORD	Returns the Item Count of the data; set to NULL if you don't need this information.
lpNumRemaining	LPLONG	A far pointer to a LONG that returns the number of DE remaining until the top is reached. If the returned value is 0, you are now at first DE, if the returned value is -1, you have attempted to move past the first DE.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

The value of lpNumRemaining tells you whether the Tag has now become the first Tag in the list.

The level_op setting determines whether the Current Data Element can move from level to level, or has to stay on the same level.

1.3.2.1.1.27 MED_DCM_DS_orig_TS_get

This function returns the original Transfer Syntax used for the image, and indicates whether of not it had a Part 10 header and the group length Data Elements.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_orig_TS_get(
        const HIGEAR hIGear,
        LPAT_MODE lpOrigTS,
        LPBOOL lpPart10,
        LPBOOL lpGrpLengths
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
lpOrigTS	LPAT_MODE	A far pointer to the original setting for TS (Transfer Syntax). Set to NULL if you do not need this information. See enumIGMedTS enumeration for complete list of Transfer Syntaxes.
lpPart10	LPBOOL	A far pointer to a BOOL which indicates whether the original file was a Part 10 file. If TRUE-the original was a Part 10 file; If FALSE-the original was not a Part 10 file. Set to NULL if you do not need this information.
lpGrpLengths	LPBOOL	A far pointer that returns a Boolean value indicating whether the original DICOM image had any Group Length Data Elements in it. TRUE means that it did have Group Length Data Elements; FALSE means that there were none. Set to NULL if you do not need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

1.3.2.1.1.28 MED_DCM_DS_part10_get

This function returns the data from the item in the Part 10 header identified by part10_item.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_part10_get(
        const HIGEAR hIGear,
        const AT_MODE part10_item,
        const DWORD size_of_lpData,
        LPVOID lpData,
        LPDWORD lpSize_of_item
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
part10_item	const AT_MODE	Set to the item in Part 10 that you would like to get. Use one of the AT_MODE constants defined in DCM.h that begin with DCM_PART10_ITEM_SET_.
size_of_lpData	const DWORD	Set to the size of the buffer lpData that you allocate to receive the data. This function will not copy more than this amount of data from the Part 10 Header, stored in the HIGEAR, to this buffer.
lpData	LPVOID	A far pointer to a VOID buffer in which return the data from the item specified by part10_item.
lpSize_of_Item	LPDWORD	A far pointer that returns the actual size of the Data Value that is stored in the Part 10 Header. It is not always the size of the data being returned in lpData. If this parameter returns a value greater than size_of_lpData then lpData does not contain all of the data-some has been clipped.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
#define BUFF_SIZE                        150
HIGEAR                  hIGear;
char                   data[BUFF_SIZE];
MED_DCM_DS_part10_get(g_hIGear, DCM_PART10_ITEM_PREAMBLE, BUFF_SIZE, data, &size_of_data);
```

1.3.2.1.1.29 MED_DCM_DS_part10_set

This function sets the data of the item in the Part 10 header identified by part10_item.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_part10_set(
        const HIGEAR hIGear,
        const AT_MODE part10_item,
        const LPVOID lpData,
        const AT_DCM_VL vl
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
part10_item	const AT_MODE	Set this variable to the type of Part 10 item you would like to set. Use one of the constants defined in DCM.h that begin with DCM_PART10_ITEM_.
lpData	const LPVOID	A far VOID pointer to the data that you would like to store into the Part 10 item.
vl	const AT_DCM_VL	Set to the length of the Data Field, in bytes.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

1.3.2.1.1.30 MED_DCM_DS_PixPadVal_get

This function retrieves the Pixel Padding Value (PPV) that is being used for the display of 16-bit grayscale images.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_PixPadVal_get(
        HIGEAR hIGear,
        LPBOOL lpUse_Pix_Padding,
        LPLONG lpPix_Padding_Val,
        LPBYTE lpShow_PPV_as
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image for which you would like to set the value of the "Pixel Padding Value" Data Element (0028,0120).
lpUse_Pix_Padding	LPBOOL	Returns TRUE if the value of Pixel Padding Value (0028,0120) will be used; FALSE if the value of Pixel Padding Value will be ignored.
lpPix_Padding_Val	LPLONG	Returns the grayscale value that will be used for padding. This value read is that which is stored in the Pixel Padding Value Data Element (0028,0120) of the internal Data Set.
lpShow_PPV_as	LPBYTE	Returns the grayscale value that will be used to display pixels equal to the Pixel Padding Value.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
AT_ERRCOUNT                nErrcount;
HIGEAR              hIGear;
LONG                Pix_Padding_Val;
BYTE                Show_PPV_as;
nErrcount = MED_DCM_DS_PixPadVal_get(g_hIGear, NULL, &Pix_Padding_Val, &Show_PPV_as);
```

Remarks:

This function also returns whether or not the PPV value will be used. The Pixel Padding Value is most often used to fill in the regions around a circular image. This function does not retrieve the PPV Data Element from the Data Set attached to the HIGEAR. It gets the value for PPV from the Internal Data Set (HDS) (which may be equal to the value in the actual Data Set). The purpose of this function is to let you know whether Pixel Padding is set on and off, what value it has, if any, and what color it is set to display as.

Please see the description for MED_DCM_DS_PixPadVal_set() for a complete description of how ImageGear handles the Pixel Padding Value Data Element.

To turn off the Pixel Padding Value or to alter the value being used use MED_DCM_DS_PixPadVal_set().

1.3.2.1.1.31 MED_DCM_DS_PixPadVal_set

This function is used to set the Pixel Padding Value that is to be used while displaying a 16-bit grayscale image.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_PixPadVal_set(
        HIGEAR hIGear,
        const BOOL Use_Pix_Padding,
        const LONG Pix_Padding_Val,
        const BYTE Show_PPV_as
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image for which you would like to set the value of the internal representation of the "Pixel Padding ValueData Element (0028,0120)".
Use_Pix_Padding	const BOOL	Set this to TRUE to use the value of Pixel Padding Value (0028,0120); FALSE to ignore the value of Pixel Padding Value.
Pix_Padding_Val	const LONG	Set this argument to the grayscale value to use for the image. This value will be stored in the Pixel Padding Value Data Element (0028,0120) of the internal Data Set. This new value is the value that will be locked into the 16x8 LUT when Pixel Padding is turned on.
Show_PPV_as	const BYTE	Set to the grayscale value that will be used to display pixels equal to the Pixel Padding Value.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
AT_ERRCOUNT              nErrcount;
HIGEAR          hIGear;
LONG            Pix_Padding_Val;
BYTE            Show_PPV_as;
BOOL            Pref_use_pix_pad;
nErrcount = MED_DCM_DS_PixPadVal_set(hIGear, bPref_use_pix_pad, Pix_Padding_Val,
Show_PPV_as);
```

Remarks:

Here is a description of what this Data Element is used for and how ImageGear implements its use:

DICOM images sometimes contain a Data Element called "Pixel Padding Value" (PPV). The PPV is used mostly to fill in the corners of round images. DICOM provides a Tag for PPV which is (0028,0120). This Data Element stores a 16-bit grayscale value that is to be treated as the Pixel Padding Value. Any pixels in the image that have this value are not to be treated as meaningful objects-but as background color.

When ImageGear Medical loads a DICOM image that contains a PPV the value is captured and stored in the HDS, which is attached to the HIGEAR of the new image. In fact, 3 values are stored to the HDS: the PPV from the PPV

Data Element, a flag indicating that a PPV was found in the file when it was loaded, and an 8-bit grayscale value to use to display pixels with this value. This function sets the values of these in-memory copies of the PPV data.

When Use_Pix_Padding is set to TRUE, pixels from the original image equal to the PPV are treated as background. All functions that fill the 16x8 LUT skip this value and place the Show_PPV_As value in the PPV slot of the table. This allows an application to adjust the contrast of the image while keeping the PPV or background constant. The background will be displayed with a grayscale value equal to that stored in Show_PPV_As. The PPV is also used for the IP functions. Functions like MED_IP_min_max() ignore pixel values that are equal to the PPV. (see below).

Use_Pix_Padding is initially set to TRUE if the loaded image contained the Pixel Padding Value Data Element (0028,0120). If this Data Element was not found then this defaults to FALSE

Pix_Padding_Val is initially set to the Data Field of the Pixel Padding Value Data Element (0028,0120) if it is found. If it is not it is set to default (NULL).

Show_PPV_As is not part of the PPV Data Element. This value always defaults to 64.

When the Pixel Padding Value is turned "on" (Use_Pix_Padding=TRUE) the MED_IP_min_max() function will know to ignore this value as it searches through the image for the brightest and darkest pixel value so that it avoids treating the Pixel Padding Value as the minimum or max pixel value. This error could easily occur because Pixel Padding Value is most often set to a very large or very small value so that it can be easily differentiated from the real pixel values.

To retrieve the Pixel Padding Value that is currently stored in the HIGEAR (not the one in the Data Set although them may have the same value) see MED_DCM_DS_PixPadVal_get().

1.3.2.1.1.32 MED_DCM_DS_preamble_get

This function gets the preamble item from the Part 10 header of the image, if one exists.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_preamble_get(
        const HIGEAR hIGear,
        LPCHAR* lpPreamble
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
lpPreamble	LPCHAR*	A far pointer to a buffer that will be used to hold the data from the preamble. The buffer must be at least 128 bytes.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

Your receiving buffer must be at least 128 bytes.

1.3.2.1.1.33 MED_DCM_DS_preamble_set

This function sets the value of the Preamble item of the Part 10 header.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_preamble_set(
        const HIGEAR hIGear,
        LPCHAR lpPreamble,
        const DWORD bytes_in_lpPreamble
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image.
lpPreamble	LPCHAR	A far pointer to the preamble of the Data Set.
bytes_in_lpPreamble	const DWORD	The number of bytes in your Preamble data.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

Provide this function with the address of the Preamble in lpPreamble, and the number of bytes for the new Preamble in bytes_in_lpPreamble. If the length of the data is less than 128 bytes, the remainder of the Preamble is filled with NULLs.

1.3.2.1.1.34 MED_DCM_DS_Rescale_get

This function will search the DICOM Data Set of the HIGEAR image and return the value from Rescale Slope (0028,1053) and Rescale Intercept (0028,1054).

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_Rescale_get(
        const HIGEAR hIGear,
        LPDOUBLE lpRescale_Slope,
        LPDOUBLE lpRescale_Intercept,
        LPBOOL lpFound
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	Set to the HIGEAR handle of the image from which you would like to retrieve the Rescale values.
lpRescale_slope	LPDOUBLE	Returns you the value of the Rescale Slope Data Element as a DOUBLE. The Tag value of this DE is (0028,1053).
lpRescale_intercept	LPDOUBLE	Returns you the value of the Rescale Intercept Data Element as a DOUBLE. The Tag value of this DE is (0028,1054).
lpFound	LPBOOL	Returns whether or not these two Data Elements were found/present in the image's Data Set.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR                  hIGear;
DOUBLE RescaleSlope, RescaleIntercept;
nErrcount = MED_DCM_DS_Rescale_get(hIGear, &RescaleSlope, &RescaleIntercept,    NULL);
```

Remarks:

This function is a short cut that was created because the values of these Data Elements are often sought after. You could perform this same operation, as you would for getting the information from any DE, by using MED_DCM_DS_move_find() and MED_DCM_DS_curr_data_get().

If both of these DEs are found in the Data Set, then lpFound returns TRUE. If one or both are missing then lpFound will return FALSE, lpRescale_slope will return 1.0, and lpRescale_intercept will return 0.0. lpFound can be set to NULL if you do not need to know if they are found or not. The returned slope and interface values returned are always usable even if they are not found in the Data Set.

Both this function and MED_DCM_DS_Window_Level_get() should be called before displaying an image using MED_display_contrast().

1.3.2.1.1.35 MED_DCM_DS_TS_get

This function returns the Transfer Syntax constant that corresponds to the value of DCM_PART10_ITEM_TRANSSYNTAXUID part 10 item in the DataSet.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_TS_get(
        const HIGEAR hIGear,
        LPAT_MODE lpTransfer_syntax
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle of the image.
lpTransfer_syntax	LPAT_MODE	Returns the Transfer Syntax constant that corresponds to the DCM_PART10_ITEM_TRANSSYNTAXUID part 10 item in the DataSet.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

lpTransfer_syntax returns one of the Transfer Syntax constants, defined in enumIGMedTS enumeration.

1.3.2.1.1.36 MED_DCM_DS_TS_set

This function sets the Transfer Syntax value in the Part 10 Header for "Transfer Syntax UID" Data Element (0002,0010).

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_TS_set(
        HIGEAR hIGear,
        const AT_MODE Transfer_syntax
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image.
Transfer_syntax	const AT_MODE	Set to the type of transfer syntax that you would like to store in the Transfer Syntax field of the Part 10 header.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This value overrides that set by MED_DCM_DS_create() function.

Transfer_syntax must be set to one of the following supported Transfer Syntax constants:

- MED_DCM_TS_IMPLICIT_VR_LE,
- MED_DCM_TS_EXPLICIT_VR_LE
- MED_DCM_TS_EXPLICIT_VR_BE

1.3.2.1.1.37 MED_DCM_DS_update_file

This function creates a new file with an exact copy of the source file's pixel data and with new metadata (File Meta Information header and DataSet) taken from HIGEAR image handle.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_update_file (
        HIGEAR hIGear,
                const LPSTR lpszFileNameSrc,
        const LPSTR lpszFileNameDst
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle containing metadata to be saved to result file.
lpszFileNameSrc	const LPSTR	Source file name.
lpszFileNameDst	const LPSTR	Result file name.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

Pixel data is not decoded, but copied directly from source to destination file.

This function can be used for updating metadata in an image file without modifying the pixel data. To achieve this, delete the source file after calling this function, and rename the result file to the source file name.

The function does not change the tags that affect decoding of pixel data.

The function takes into account the following DICOM filter control parameters:

- SAVE_GROUPLENGTHS
- SAVE_ASPART10

1.3.2.1.1.38 MED_DCM_DS_update_from

This function copies the Element List associated with hSrcList to the image associated with hIGear.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_update_from(
        HIGEAR hIGear,
        HIGMEDELEMLIST hSrcList
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to an image, from which the Element List will be copied.
hSrcList	HIGMEDELEMLIST	HIGMEDELEMLIST handle to the Data Element List object.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

1.3.2.1.1.39 MED_DCM_DS_Window_Level_get

This function searches the Data Set of the HIGEAR image for the Window Width and Window Center Data Elements.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_Window_Level_get(
        const HIGEAR hIGear,
        LPLONG lpWindow_Width,
        LPLONG lpWindow_Center,
        LPBOOL lpFound
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image from which you would like to retrieve the Window Level values.
lpWindow_Width	LPLONG	Returns the value of the Window Level Width Data Element (0028,1051).
lpWindow_Center	LPLONG	Returns the value of the Window Level Center Data Element (0028,1050).
lpFound	LPBOOL	Returns TRUE if both of these DEs are found in the Data Set; returns FALSE if one or the other is missing from the Data Set. Set to NULL if you do not need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
AT_ERRCOUNT              nErrcount;
HIGEAR           hIGear;
LONG             lWindow_min, lWindow_max;
nErrcount = MED_DCM_DS_Window_Level_get(hIGear, &lWindow_min, &lWindow_max,     NULL);
```

Remarks:

This function is a shortcut that was created because the values of these Data Elements are often sought after. You could perform this same operation, as you would for getting the information from any DE, by using MED_DCM_DS_move_find() and MED_DCM_DS_curr_data_get().

If both are found they are returned and lpFound is set to TRUE. If one or both of these DEs are not found, lpFound is set to FALSE and ImageGear attempts to calculate adequate values for both Width and Center to display all pixels in the image.

For 17-32 bits per pixel images, please use MED_DCM_DS_Window_Level_get_64().

1.3.2.1.1.40 MED_DCM_DS_Window_Level_get_64

This function searches the Data Set of the HIGEAR image for the Window Width and Window Center Data Elements.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_DS_Window_Level_get_64(
        const HIGEAR hIGear,
        LPAT_INT64 lpWindow_Width,
        LPAT_INT64 lpWindow_Center,
        LPBOOL lpFound
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	HIGEAR handle to the image from which you would like to retrieve the Window Level values.
lpWindow_Width	LPAT_INT64	Returns the value of the Window Level Width Data Element (0028,1051) as 64 bit integer.
lpWindow_Center	LPAT_INT64	Returns the value of the Window Level Center Data Element (0028,1050) as 64 bit integer.
lpFound	LPBOOL	Returns TRUE if both of these DEs are found in the Data Set; returns FALSE if one or the other is missing from the Data Set. Set to NULL if you do not need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
AT_ERRCOUNT              nErrcount;
HIGEAR          hIGear;
AT_INT64                lWindow_min, lWindow_max;
nErrcount = MED_DCM_DS_Window_Level_get_64(hIGear, &lWindow_min, &lWindow_max,  NULL);
```

Remarks:

Use this function for 17-32 bits per pixel images. Although you can use this function for 8-16 bit images, it may impact performance in 32 bit operation systems. This function is a shortcut that was created because the values of these Data Elements are often sought after. You could perform this same operation, as you would for getting the information from any DE, by using MED_DCM_DS_move_find() and MED_DCM_DS_curr_data_get().

If both are found they are returned and lpFound is set to TRUE. If one or both of these DEs are not found, lpFound is set to FALSE and ImageGear attempts to calculate adequate values for both Width and Center to display all pixels in the image.

1.3.2.1.2 Display Functions

This section provides information about the Display group of functions.

- MED_display_color_create
- MED_display_color_limits
- MED_display_color_set
- MED_display_contrast
- MED_display_contrast_auto
- MED_display_grayscale_LUT_build
- MED_display_grayscale_LUT_build_auto
- MED_VOI_window_init_from_min_max
- MED_VOI_window_max_get
- MED_VOI_window_min_get

1.3.2.1.2.1 MED_display_color_create

This function is used to fill 3 LUTs with one of several pseudo-color schemes.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_color_create(
        const AT_MODE scheme,
        const LONG param1,
        const LONG param2,
        const LONG param3,
        LPAT_PIXEL lpRLUT,
        LPAT_PIXEL lpGLUT,
        LPAT_PIXEL lpBLUT
);
```

Arguments:

Name	Type	Description
scheme	const AT_MODE	Set to the pre-defined color scheme that you would like to use to pseudo-color an image. These AT_MODE color schemes are defined in MedAPI.h and begin with DCM_PSEUDOCOLOR_SCHEME_ . See below for details.
param1	const LONG	Set to a LONG value if the color scheme requires it. See below.
param2	const LONG	Set to a LONG value if the color scheme requires it. See below.
param3	const LONG	Set to a LONG value if the color scheme requires it. See below.
lpRLUT	LPAT_PIXEL	Pass this argument an array of 256 bytes. It returns the Red LUT that will be used to pseudo-color an image when MED_display_color_set() is called.
lpGLUT	LPAT_PIXEL	Pass this argument an array of 256 bytes. It returns the Green LUT that will be used to pseudo-color an image when MED_display_color_set() is called.
lpBLUT	LPAT_PIXEL	Pass this argument an array of 256 bytes. It returns the Blue LUT that will be used to pseudo-color an image when MED_display_color_set() is called.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIGEAR          hIGear;
{
AT_PIXEL                RLUT[256];
AT_PIXEL                 GLUT[256];
AT_PIXEL                 BLUT[256];
MED_display_color_create(DCM_PSEUDOCOLOR_SCHEME_6, 0, 0, 0, RLUT, GLUT, BLUT);
MED_display_color_set(hIGear, RLUT, GLUT, BLUT);
repaint_image_and_error_check(hWnd);
}
```

Remarks:

The scheme parameter selects which ImageGear pre-defined pseudo-color method to use. Some of the schemes may require parameters that are passed into param1, param2, and param3. Other schemes do not use these parameters and any value passed in are ignored. See below.

lpRLUT, lpGLUT, and lpBLUT must be passed an array of 256 bytes, each. When this function returns, these 3 tables will be filled with values that can be used to pseudo-color an 8-16 bit grayscale image.

This function does not apply the color to an image. It only fills these 3 LUTs. To apply these tables to an image you need to follow this function call with a call to MED_display_color_set().

Currently, these are the available preset ImageGear pseudo-color schemes:

DCM_PSEUDOCOLOR_SCHEME_OFF	Reset to grayscale - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_1	Oil Film - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_2	Dark Blue to Bright Red - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_3	Green to Red - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_4	Red, Green, Blue - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_5	Thermal - does not use param1, param2, param3
DCM_PSEUDOCOLOR_SCHEME_6	Bright Rainbow - does not use param1, param2, param3

1.3.2.1.2.2 MED_display_color_limits

This function can be used to pseudo-color the brightest and the darkest pixels in a 16-bit grayscale image.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_color_limits(
        HIGEAR hIGear,
        const INT thresh_low,
        const INT low_red,
        const INT low_green,
        const INT low_blue,
        const INT thresh_high,
        const INT high_red,
        const INT high_green,
        const INT high_blue
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
thresh_low	const INT	Set to an INT value for the lower pixel value limit. All pixels at or below this value will be pseudo-colored.
low_red	const INT	Set to an INT for the red component of the RGB color that will be used for all pixels below thresh_low.
low_green	const INT	Set to an INT for the green component of the RGB color that will be used for all pixels below thresh_low.
low_blue	const INT	Set to an INT for the blue component of the RGB color that will be used for all pixels below thresh_low.
thresh_high	const INT	Set to an INT value for the upper pixel value limit. All pixels at or above this value will be pseudo-colored.
high_red	const INT	Set to an INT for the red component of the RGB color that will be used for all pixels above thresh_high.
high_green	const INT	Set to an INT for the green component of the RGB color that will be used for all pixels above thresh_high.
high_blue	const INT	Set to an INT for the blue component of the RGB color that will be used for all pixels above thresh_high.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

This is typically used to clearly see pixel values that are over-saturated (255 and up) or under-saturated (0 and below). However, this function lets you customize the settings for the upper and lower pixel value limits with thresh_low and thresh_high.

The color is applied after the 16x8 display LUT is applied to the image and is not affected by altering this LUT.

Pixels having values at or below thresh_low are colored by low_color and pixels having values at or above thresh_high are colored by high_color. All pixels having values within the 2 limits are set to normal linear gray.

To turn off this effect, set thresh_low to -1 and thresh_high to 256.

This function does not cause the image to be displayed or repainted. It only fills the 16x8 LUT. To display the results of this function, use IG_dspl_image_draw().

1.3.2.1.2.3 MED_display_color_set

This function allows applications to apply pseudo-color to 8-16-bit grayscale images.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_color_set(
        HIGEAR hIGear,
        const DWORD dwGrpID,
        const LPAT_PIXEL lpRLUT,
        const LPAT_PIXEL lpGLUT,
        const LPAT_PIXEL lpBLUT
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the 8-16-bit grayscale image to color.
dwGrpID	const DWORD	Display group identifier that should be used for display operations.
lpRLUT	const LPAT_PIXEL	Set to a 256 byte Red table or set to NULL for ramp.
lpGLUT	const LPAT_PIXEL	Set to a 256 byte Green table or set to NULL for ramp.
lpBLUT	const LPAT_PIXEL	Set to a 256 byte Blue table or set to NULL for ramp.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Example:

```
AT_ERRCOUNT nErrcount;
HIGEAR          hIGear;
{
AT_PIXEL                RLUT[256];
AT_PIXEL                GLUT[256];
AT_PIXEL                BLUT[256];
MED_display_color_create(DCM_PSEUDOCOLOR_SCHEME_6, 0, 0, 0, RLUT, GLUT, BLUT);
MED_display_color_set(hIGear, 0, RLUT, GLUT, BLUT);
repaint_image_and_error_check(hWnd);
}
```

Remarks:

The 3 LUTs can be user-defined or they can be filled by calling MED_display_color_create(). Each LUT must either be 256 bytes each or set to NULL. If a NULL is passed in as one of the LUTs, a linear 0-255 grayscale LUT is used. If a NULL is not passed in, the LUT points to an array of 256 bytes. The entries from this LUT are used to color the output of the 16x8 LUT.

The pseudo-coloring is not altered by any functions that update the 16x8 LUT. This function alters the display of a grayscale image only and does not change the pixel values of the image or any entries in the 16x8 LUT.

To turn off the pseudo-color, simply pass in NULLs for all of the LUT parameters. These three color channels will be reset to their default linear ramps.

This function does not cause the image to be displayed or repainted. It only fills the 16x8 LUT. To display the results of this function, use IG_dspl_image_draw().

1.3.2.1.2.4 MED_display_contrast

This function uses the window/level mapping method to change the contrast of an 9-16-bit image.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_contrast(
        HIGEAR hIGear,
        const DOUBLE rescale_slope,
        const DOUBLE rescale_intercept,
        const LONG window_center,
        const LONG window_width,
        const DOUBLE gamma
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	16g image having its 16x8 LUT updated.
rescale_slope	const DOUBLE	Set to the desired value for Rescale Slope (0028,1053). Call MED_DCM_DS_Rescale_get() to obtain this value.
rescale_intercept	const DOUBLE	Set to the desired value for Rescale Intercept (0028,1054). Call MED_DCM_DS_Rescale_get() to obtain this value.
window_center	const LONG	Set to the desired value for Window Center (0028,1050). Call MED_DCM_DS_Window_Level_get() to obtain this value.
window_width	const LONG	Set to the desired value for Window Width (0028,1051). Call MED_DCM_DS_Window_Level_get() to obtain this value.
gamma	const DOUBLE	Set this to the amount of Gamma correction you would like applied to the image. To turn off Gamma correction, set to 1.0. The valid range of values is any DOUBLE between 0.20-1.80.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…32 bpp.

Remarks:

It takes values for Window Center & Width, and Rescale Slope & Intercept, and it also allows you to apply Gamma correction, if you desire.

This function fills the 16x8 LUT with values that will display a 9-16 bit image according to VOI LUT (Window Center/Width) and Modality LUT (Rescale Intercept/Slope) values that you specify. Any values that have been in the 16x8 LUT are overwritten with these new ones.

If the Rescale Slope and Intercept of the image are known, you should provide them. If they are not known, pass in a 1.0 and 0.0, respectively.

If you do not have values for Window Center and Window Width, these can be calculated using the minimum and maximum pixel values (if you have them known). Here are the formulas you should use to display all pixel values:

Window Center = (max + min) / 2;

Window Width = (max - min);

This function does not cause the image to be displayed or repainted. It only fills the 16x8 LUT. To display the results of this function, use IG_dspl_image_draw().

See also MED_display_contrast_auto()

1.3.2.1.2.5 MED_display_contrast_auto

This function automatically fills the 16x8 LUT of a 16-bit image loaded into ImageGear in order to optimize its displaying.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_contrast_auto(
        const HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const DOUBLE rescale_slope,
        const DOUBLE rescale_intercept,
        const DOUBLE gamma,
        const LONG lReserved_option,
        LPLONG lpWindow_center,
        LPLONG lpWindow_width
);
```

Arguments:

Name	**Type**	**Description**
hIGear	const HIGEAR	The HIGEAR handle to the image to convert.
lpRect	const AT_RECT	Use this AT_RECT structure to specify the rectangular portion of the image for which to optimize the contrast on; set to NULL for the whole image. Please see the ImageGear User's Manual if you are unfamiliar with this structure.
rescale_slope	const DOUBLE	Set to the value of the Data Element, Rescale Slope (0028,1053). You can use MED_DCM_DS_Rescale_get() to obtain this value. If this Data Element is not present in the Data Set, please set this value to 1.0.
rescale_intercept	const DOUBLE	Set to the value of the Data Element, Rescale Intercept (0028,1054). You can use MED_DCM_DS_Rescale_get() to obtain this value. If this Data Element is not present in the Data Set, please set this value to 0.0.
gamma	DOUBLE	Gamma correction for 16x8 Lookup Table. Set to 1.0 to turn the correction off.
lReserved_option	const LONG	Reserved for future use; please set to 0 for now.
lpWindow_center	LPLONG	A far pointer that returns a LONG for the Window Center; set to NULL if you don't need this information.
lpWindow_width	LPLONG	A far pointer that returns a LONG for the Window Width; set to NULL if you don't need this information.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…32 bpp.

Example:

```
HIGEAR hIGear;
DOUBLE          fRescaleSlope = 1.0;
DOUBLE          fRescaleIntercept = 1.0;
DOUBLE          fGama = 1.0;
LONG            WindowWidth;
LONG            WindowCenter;
MED_display_contrast_auto(hIGear, NULL, fRescaleSlope,
                         fRescaleIntercept, fGamma, 0, &WindowCenter,
                                                       &WindowWidth);
```

Remarks:

The rectangular portion of the image that you specify (or the whole image) is scanned and the maximum and minimum pixel values are determined; the Window Center and Width are calculated from these.

If the Rescale Slope and Intercept of the image are known, you should provide them. If they are not known, pass in a 1.0 and 0.0, respectively.

The Window Center and Width that this function computes are passed back to you in lpWindow_center and lpWindow_width, unless you set these arguments to NULL.

This function does not cause the image to be displayed or repainted. It only fills the 16x8 LUT. To display the results of this function, use IG_dspl_image_draw().

See also MED_display_contrast().

Since this function must scan the pixels for the minimum and maximum values, it takes more time to run than IG_display_contrast().

1.3.2.1.2.6 MED_display_grayscale_LUT_build

This function fills a grayscale LUT according to lpDICOMDisplaySettings.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_grayscale_LUT_build(
        AT_MED_DCM_DISPLAY_SETTINGS* lpDICOMDisplaySettings,
        HIGLUT hLUT
);
```

Arguments:

Name	Type	Description
lpDICOMDisplaySettings	AT_MED_DCM_DISPLAY_SETTINGS*	AT_MED_DCM_DISPLAY_SETTINGS structure.
hLUT	HIGLUT	LUT handle.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The LUT must be created prior to calling this function. Use IG_LUT_create() to create a LUT.

1.3.2.1.2.7 MED_display_grayscale_LUT_build_auto

This function calculates lpDICOMDisplaySettings->VOIWindow from image's min and max values, and then builds grayscale LUT according to lpDICOMDisplaySettings.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_display_grayscale_LUT_build_auto(
        HIGEAR hIGear,
        const AT_RECT* lpRect,
        AT_MED_DCM_DISPLAY_SETTINGS* lpDICOMDisplaySettings,
        HIGLUT hlut
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	16g image having its 16x8 LUT updated.
lpRect	const AT_RECT*	AT_RECT structure.
lpDICOMDisplaySettings	AT_MED_DCM_DISPLAY_SETTINGS*	AT_MED_DCM_DISPLAY_SETTINGS structure.
hlut	HIGLUT	LUT handle.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…32 bpp.

1.3.2.1.2.8 MED_VOI_window_init_from_min_max

This function initializes a AT_MED_VOI_WINDOW structure from window Min and Max.

Declaration:

```
AT_VOID MED_VOI_window_init_from_min_max(
        AT_MED_VOI_WINDOW* lpWindow,
        AT_INT Min,
        AT_INT Max
);
```

Arguments:

Name	Type	Description
lpWindow	AT_MED_VOI_WINDOW*	Window structure.
Min	AT_INT	Window min.
Max	AT_INT	Window max.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

1.3.2.1.2.9 MED_VOI_window_max_get

This function returns AT_MED_VOI_WINDOW maximum.

Declaration:

```
AT_INT MED_VOI_window_max_get(
        AT_MED_VOI_WINDOW* lpWindow
);
```

Arguments:

Name	Type	Description
lpWindow	AT_MED_VOI_WINDOW*	Window structure.

Return Value:

AT_MED_VOI_WINDOW maximum.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.2.1.2.10 MED_VOI_window_min_get

This function returns AT_MED_VOI_WINDOW minimum.

Declaration:

```
AT_INT MED_VOI_window_min_get(
        AT_MED_VOI_WINDOW* lpWindow
);
```

Arguments:

Name	Type	Description
lpWindow	AT_MED_VOI_WINDOW*	Window structure.

Return Value:

AT_MED_VOI_WINDOW minimum.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.2.1.3 File Functions

This section provides information about the File group of functions.

- MED_DCM_load_DICOM
- MED_DCM_load_DICOM_FD
- MED_DCM_save_DICOM
- MED_DCM_save_DICOM_FD

1.3.2.1.3.1 MED_DCM_load_DICOM

This function loads a DICOM image file.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_load_DICOM(
        const LPSTR lpszFileName,
        LPHIGEAR lphIGear,
        const AT_MODE nSyntax,
        const UINT page_number
);
```

Arguments:

Name	Type	Description
lpFileName	const LPSTR	A far pointer to the name of the file to load.
lphIGear	LPHIGEAR	A far pointer which returns the HIGEAR handle for the newly loaded image.
nSyntax	const AT_MODE	Set to the type of Transfer Syntax used for the file that you will be loading. Use one of the MED_DCM_TS_ constants defined in enumIGMedTS.
page_number	const UINT	If the file is a multi-page (multiframe) file, you may set this variable to specify the page to load. If the file is not multi-page, set this to 1.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional DICOM format filter.

Remarks:

This function cannot be used to load an image of any other format. The advantage to using this function over IG_load_file() is that it provides more control, and is a faster loading utility because ImageGear's filter detection is not used.

This function loads the file specified by the path and filename in lpFileName, and returns a new HIGEAR handle for the image in lphIGear.

To further speed up this function, provide the correct Transfer Syntax (TS) of the file. If you do not know the TS, ImageGear detects it for you if you set nSyntax to MED_DCM_TS_AUTODETECT.

You may also specify the page to load from a multi-page (multiframe) file.

To remove a HIGEAR image from memory, call IG_image_delete().

If your application supports unicode or multi-byte strings, you can open a file yourself and pass the FD handle. If you prefer to open your own file, use MED_DCM_load_DICOM_FD() .

1.3.2.1.3.2 MED_DCM_load_DICOM_FD

This function performs the same operation as MED_DCM_load_DICOM() except that is takes a File Descriptor (FD) and offset instead of a filename.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_load_DICOM_FD(
        const AT_INT fd,
        const LONG lOffset,
        LPHIGEAR lphIGear,
        const AT_MODE nSyntax,
        const UINT page_number
);
```

Arguments:

Name	Type	Description
fd	const AT_INT	Set to the File Descriptor handle of the opened image file.
lOffset	const LONG	Set to a LONG value that gives the offset from the current position in the open file to the beginning of the DICOM data. In most cases this value will be 0.
lphIGear	LPHIGEAR	A far pointer which returns the HIGEAR handle for the newly loaded image.
nSyntax	const AT_MODE	Set to the type of Transfer Syntax used for the file that you will be loading. Use one of the MED_DCM_TS_ constants defined in enumIGMedTS.
page_number	const UINT	If the file is a multi-page file, you may set this variable to specify the page to load. If the file is not multi-page, set this to 1.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional DICOM format filter.

Remarks:

lOffset should be set to the position of the image within the file so that the ImageGear Medical knows where to begin decoding the image. If the beginning of the DICOM image file starts at the first byte, pass in an 0L for lOffset.

Since you are responsible for opening and closing the file, this function can be used to bypass the IG_load_file() limitation of not handling unicode or multi-byte character strings.

1.3.2.1.3.3 MED_DCM_save_DICOM

This function saves a DICOM image file.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_save_DICOM(
        const LPSTR lpszFileName,
        const HIGEAR hIGear,
        const AT_MODE nSyntax,
        const BOOL bIncludeGroupLengths,
        const BOOL bSaveAsPart10,
        const AT_MODE PlanarConfiguration,
        const BOOL IncludeSmallestLargest,
        const UINT nJPEGQuality,
        const DWORD dwReserved
);
```

Arguments:

Name	Type	Description
lpszFileName	const LPSTR	A far pointer to the name of the file to save.
hIGear	const HIGEAR	The HIGEAR handle of the image to save. This HIGEAR must have an attached DICOM Data Set.
nSyntax	const AT_MODE	Set to the type of Transfer Syntax with which to save the image.
bIncludeGroupLengths	const BOOL	Set this Boolean variable to TRUE if you want to store the Group Lengths with the image.
bSaveAsPart10	const BOOL	Set this Boolean variable to TRUE if you want to save a Part 10 header with the image.
PlanarConfiguration	const AT_MODE	Determines how RGB pixel values are to be saved to the DICOM image file. The Planar Configuration Data Element (0028, 0006) will be automatically inserted and this value stored in it. If the Data Set already contains (0028,006) its value is ignored and the value of this parameter is used instead. This parameter is used for RGB only. It is ignored for all other image types. Set to one of the constants that begins with MED_DCM_PLANAR_.
IncludeSmallestLargest	const BOOL	If the image's original Data Set did not contain Data Elements for Smallest Image Pixel Value (0028,0106) and Largest Image Pixel Value (0028,0107) and you set IncludeSmallestLargest = TRUE, ImageGear will scan the image and determine a values for these DEs. Smallest Image Pixel Value and Largest Image Pixel Value will be included in the Data Set of the DICOM image being saved, and will contain the ImageGear-determined values. The values of the DE from the original Data Set (if any) will be ignored. If you set IncludeSmallestLargest = FALSE, ImageGear will not determine this value for you, and the Data Set of the image being saved will not include the Smallest Image Pixel Value and Largest Image Pixel Value Data Elements. However, if the original Data Set did contain these DEs, ImageGear will preserve and include them in the Data Set being saved.
nJPEGQuality	const UINT	The value of this argument is only relevant to lossy JPEG compression. When using any other compression scheme, this value will be ignored. (This setting would be meaningless for Lossless JPEG compression). Set to the amount of pixel data to preserve during lossy JPEG compression. The range of valid values is 1-100 with a default value of 70. Higher settings result in higher quality and a larger file. Note that even at 100, JPEG compression is not capable of being completely "lossless." The compression used, if any, is set by nSyntax. (MED_DCM_TS_JPEG_LOSSY_8 indicates the use of 8-bit JPEG compression).

dwReserved	const DWORD	Reserved for future use. Set to 0 for now.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This function cannot be used to save an image of any other format. It provides a convenient way to save DICOM images, using DICOM-specific saving options. However, general saving functions such as IG_fltr_save_file, together with DICOM control parameters, provide greater flexibility for saving DICOM images. We recommend that you use the latter way of saving DICOM images.

In order to save an image as a DICOM file the HIGEAR must have a Data Set attached to it. If you loaded the HIGEAR from a DICOM file then there already is a Data Set attached. If not, you can use the function MED_DCM_DS_create() to create one. If you are not sure if the image has a Data Set, call MED_DCM_DS_exists() to find out. If you are going to create a Data Set using MED_DCM_DS_create(), then you must still populate the Data Set with valid Data Elements using MED_DCM_DS_DE_insert().

The nSyntax parameter determines how the DICOM image file is encoded. The valid options are the constants whose names begin with MED_DCM_TS_ (except for _TS_UNKNOWN, and _TS_AUTODETECT).

Group Lengths are optional in a DICOM image file. By default, Group Lengths are not stored in the Data Set that is attached to the HIGEAR. If you have read a DICOM file that included Group Lengths, they have been discarded as they were found. Set bIncludeGroupLength to TRUE to have them recreated and placed in the Data Set as it has been written to disk.

DICOM Image files are supposed to be written to disk using the Meta-Info Header that is defined in Part 10 of the DICOM Specification. However, many DICOM applications choose not to use this header. If bSaveAsPart10 is set to TRUE, the Meta Information Header will be placed at the beginning of the file. Setting this to FALSE will skip the header.

If the image is saved without a Part 10 header, it is often called a Raw DICOM image file. Note that the Part 10 Header Data Elements are not stored the same way as other Data Elements in the Data Set. If you wish to store this data you must populate the header using the MED_DCM_DS_part10_set() function before writing the file.

If your application supports unicode or multi-byte strings, you can open a file yourself and pass us the FD handle. If you prefer to open your own file, use MED_DCM_save_DICOM_FD().

1.3.2.1.3.4 MED_DCM_save_DICOM_FD

This function performs the same operation as MED_DCM_save_DICOM() except that it uses the FD instead of the file name.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_save_DICOM_FD(
        AT_INT fd,
        const HIGEAR hIGear,
        const AT_MODE nSyntax,
        const BOOL bIncludeGroupLengths,
        const BOOL bSaveAsPart10,
        const AT_MODE nPlanarConfiguration,
        const BOOL bInludeSmallestLargest,
        const UINT nJPEGQuality,
        const DWORD lReserved
);
```

Arguments:

Name	Type	Description
fd	AT_INT	Set to the FD handle of the open image. The image must have been opened with Read/Write access.
hIGear	const HIGEAR	The HIGEAR handle of the image to save. This HIGEAR must have an attached DICOM Data Set.
nSyntax	const AT_MODE	Set to the type of Transfer Syntax used for the file that you will be saving. Use one of the MED_DCM_TS_ constants defined in enumIGMedTS.
bIncludeGroupLengths	const BOOL	Set this Boolean variable to TRUE if you want to store the Group Lengths with the image.
bSaveAsPart10	const BOOL	Set this Boolean variable to TRUE if you want to save a Part 10 header with the image.
nPlanarConfiguration	const AT_MODE	Determines how RGB pixel values are to be saved to the DICOM image file. The Planar Configuration Data Element (0028, 0006) will be automatically inserted and this value stored in it. If the Data Set already contains (0028,006), its value is ignored, and the value of this parameter is used instead. This parameter is used for RGB only. It is ignored for all other image types. Set to one of the constants that begins with MED_DCM_PLANAR_.
bIncludeSmallestLargest	const BOOL	Setting this parameter to TRUE will cause the Smallest and Largest pixel value in the image to be computed and saved in Smallest Image Pixel Value (0028,0106) and Largest Image Pixel Value (0028,0107). If the Data Set contains these 2 Data Elements already, they are ignored and the computed values are used. If this parameter is set to FALSE these two Data Elements are not stored in the file.
nJPEGQuality	const UINT	The value of this argument is only relevant to lossy JPEG compression. When using any other compression scheme, this value will be ignored. Set to the amount of pixel data to preserve during lossy JPEG compression. The range of valid values is 1-100 with a default value of 70. Higher settings result is higher quality and a larger file. Note that even at 100, JPEG compression is not capable of being completely "lossless".
lReserved	const DWORD	Reserved for future use. Set to 0 for now.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by IG_FORMAT_DCM format.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

The file must have been opened with Read/Write access.

Since you are responsible for opening and closing the file, this function can be used to bypass the IG_save_file() limitation of not handling unicode or multi-byte character strings.

1.3.2.1.4 Image Processing Functions

This section provides information about the Image Processing group of functions.

- MED_IP_contrast
- MED_IP_contrast_auto
- MED_IP_high_bit_transform
- MED_IP_histo_clear
- MED_IP_histo_tabulate
- MED_IP_min_max
- MED_IP_min_max_64
- MED_IP_normalize
- MED_IP_promote_to_16_gray
- MED_IP_reduce_depth_with_downshift
- MED_IP_reduce_depth_with_LUT
- MED_IP_swap_bytes

1.3.2.1.4.1 MED_IP_contrast

This function converts a 16-bit grayscale image to an 8-bit grayscale image in the same way as MED_display_contrast().

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_contrast(
        HIGEAR hIGear,
        const DOUBLE rescale_slope,
        const DOUBLE rescale_intercept,
        const LONG window_center,
        const LONG window_width,
        const DOUBLE gamma
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	16g image that will have its 16x8 LUT updated.
rescale_slope	const DOUBLE	Set to the desired value for Rescale Slope (0028,1053). Call MED_DCM_DS_Rescale_get() to obtain this value.
rescale_intercept	const DOUBLE	Set to the desired value for Rescale Intercept (0028,1054). Call MED_DCM_DS_Rescale_get() to obtain this value.
window_center	const LONG	Set to the desired value for Window Center (0028,1050). Call MED_DCM_DS_Window_Level_get() to obtain this value.
window_width	const LONG	Set to the desired value for Window Width (0028,1051). Call MED_DCM_DS_Window_Level_get() to obtain this value.
gamma	const DOUBLE	Set this to the amount of Gamma correction you would like applied to the image. To turn off Gamma correction, set to 1.0. The valid range of values is any DOUBLE between 0.20-1.80.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 9…16 bpp.

Remarks:

The difference between this function and MED_display_contrast() is that this function permanently alters the pixel values.

See also MED_display_contrast().

The functionality of MED_IP_window_level() has been incorporated into the new function MED_IP_contrast(), which also takes settings for Rescale Slope, Rescale Intercept, and gamma correction.

1.3.2.1.4.2 MED_IP_contrast_auto

This function converts a 16-bit grayscale image to an 8-bit grayscale image using the same function as MED_display_contrast_auto() except that this function permanently alters the pixel values.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_contrast_auto(
        const HIGEAR hIGear,
        const LPAT_RECT lpRect,
        const DOUBLE rescale_slope,
        const DOUBLE rescale_intercept,
        const DOUBLE gamma,
        const LONG lReserved_option,
        LPLONG lpWindow_center,
        LPLONG lpWindow_width
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	The HIGEAR handle to the image to convert.
lpRect	const AT_RECT	Use this AT_RECT structure to specify the rectangular portion of the image for which to optimize the contrast on; set to NULL for the whole image. Please see the ImageGear User's Manual if you are unfamiliar with this structure.
rescale_slope	const DOUBLE	Set to the value of the Data Element, Rescale Slope (0028,1053). You can use MED_DCM_DS_Rescale_get() to obtain this value. If this Data Element is not present in the Data Set, please set this value to 1.0.
rescale_intercept	const DOUBLE	Set to the value of the Data Element, Rescale Intercept (0028,1054). You can use MED_DCM_DS_Rescale_get() to obtain this value. If this Data Element is not present in the Data Set, please set this value to 0.0.
gamma	const DOUBLE	Non-linear method to adjust the contrast of DICOM image. In this method, the amount a pixel's intensity changes depends on its original intensity. Usual range is 0.75 to 3.0.
lReserved_option	const LONG	Reserved for future use; please set to 0 for now.
lpWindow_center	LPLONG	A far pointer that returns a LONG for the Window Center; set to NULL if you don't need this information.
lpWindow_width	LPLONG	A far pointer that returns a LONG for the Window Width; set to NULL if you don't need this information.

Supported Raster Image Formats:

Grayscale – 9…16 bpp.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

See Also

MED_display_contrast_auto()

MED_IP_reduce_depth_with_downshift()

1.3.2.1.4.3 MED_IP_high_bit_transform

This function changes the High Bit Data Element of the currently loaded DICOM 16-bit grayscale image.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_high_bit_transform(
        HIGEAR hIGear,
        const LONG lMin
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image.
lMin	const LONG	New High bit value.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 9…16 bpp.

Remarks:

The 16-bits are shifted up or down as needed to accommodate the new High Bit value, and the High Bit Data Element is updated. Zeros are shifted in as needed.

1.3.2.1.4.4 MED_IP_histo_clear

This function is used to clear the histogram created by MED_IP_histo_tabulate().

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_histo_clear(
        LPLONG lpHisto,
        const DWORD nBin_count
);
```

Arguments:

Name	Type	Description
lpHisto	LPLONG	A far pointer of type LONG which points to the buffer that will was used to hold the histogram.
nBin_count	const DWORD	Set this DWORD variable to the number of bins allocated for the histogram.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.2.1.4.5 MED_IP_histo_tabulate

This function is used to tabulate the histogram of a 8 or 16-bit grayscale image.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_histo_tabulate(
        const HIGEAR hIGear,
        const LPAT_RECT lpRect,
        LPLONG lpHisto,
        const WORD nBin_width,
        const DWORD dwBin_count,
        LPBOOL lpSigned,
        LPLONG lpCount
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	The HIGEAR handle to the image for which to create a histogram.
lpRect	const LPAT_RECT	A far pointer to a struct of type AT_RECT that defines the rectangular portion of the image to use for creating a histogram. Please see the ImageGear User's Manual if you are unfamiliar with this structure.
lpHisto	LPLONG	A far pointer to a buffer to be used for holding the histogram.
nBin_width	const WORD	An integer variable that specifies the range of pixel values to be counted into one bin.
dwBin_count	const DWORD	A DWORD variable that specifies the number of bins allocated.
lpSigned	LPBOOL	A far pointer to a BOOL that returns the sign status of the image. If it returns TRUE, the image is signed.
lpCount	LPLONG	A far pointer to a LONG which returns the number of pixels counted.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…16 bpp.

Remarks:

The histogram is returned in the memory block, lpHisto, allocated by the application. This memory block must be large enough to hold the histogram for the image. Each histogram bin must be 4 bytes wide. The size of lpHisto needed (in bytes) can be computed as follows:

size = ((possible_pixel_values) * 4) / nBin_width

where possible_pixel_values depends on the bit depth of the image (8g=256, 9g=512,...). nBin_width is used to determine the range of pixel values that are counted in their own bin. A value of 1 indicates that each pixel value is counted in its own histogram bin. A value of 2 would allow neighboring values (such as 128 and 129) to be counted as a single bin (as a single value). If nBin_width==possible_pixel_values then only a single histogram bin (4 bytes) is filled and the count will be equal to the number of pixel values in the lpRect.

nBin_count is used as a safety. This should be set to the number of bins in the histogram that your application has allocated. If a pixel value is going to overflow this memory it will be ignored. This pixel value will not be included in the sum returned in lpCount (you can use this to determine if any values were ignored).

1.3.2.1.4.6 MED_IP_min_max

This function scans an 8- or 16-bit grayscale image and returns the raw minimum and maximum pixel values and the "is signed" flag.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_min_max(
        const HIGEAR hIGear,
        const LPAT_RECT lpRect,
        LPLONG lpMin,
        LPLONG lpMax,
        LPBOOL lpSigned
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	The HIGEAR handle of the image to scan.
lpRect	const AT_RECT	Use this AT_RECT structure to specify the rectangular portion of the image to scan. Please see the ImageGear User's Manual if you are unfamiliar with this structure.
lpMin	LPLONG	A far pointer to a LONG which returns the value of the minimum pixel value in the region that was "scanned".
lpMax	LPLONG	A far pointer to a LONG which returns the value of the maximum pixel value in the region that was "scanned".
lpSigned	LPBOOL	A far pointer to a BOOL which returns the status of the image sign. TRUE means the image is signed.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…16 bpp.

Example:

```
HIGEAR hIGear;
AT_ERRCOUNT nErrcount;
AT_RECT rcROI;
LONG lMin, lMax;
BOOL bSigned;
nErrcount = MED_IP_min_max(hIGear, &rcROI, &lMin, &lMax, &bSigned);
```

Remarks:

If the image is 16-bit signed, then the returned values are also signed.

Note that this function returns the min and max raw pixel values. That is, the returned values are not corrected using the Modality LUT (Rescale Slope/Intercept) values. To apply this correction to the min and max values that you get from MED_IP_min_max() call the function MED_DCM_DS_Rescale_get() which will return you the values of Rescale Slope and Rescale Intercept. Then use the following formulas:

min_corrected = (min_raw * rescale_slope) + rescale_intercept;

max_corrected = (max_raw * rescale_slope) + rescale_intercept;

When a 16-bit grayscale image is "unsigned" it has pixel values between 0 and 65,000. If a 16-bit image is "signed" it has pixel values between -32k and +32k. By the same rule, when an 8-bit grayscale image is "unsigned" it has pixel values between 0-255. If an 8-bit image is "signed", it has pixel values between -128 and +127. Some modalities of DICOM use signed images. So if you know whether the image is signed or unsigned will help you to interpret the minimum and maximum values.

For 17-32 bits per pixel images, please use MED_IP_min_max_64().

1.3.2.1.4.7 MED_IP_min_max_64

This function scans an 17-32 bit grayscale image and returns the raw minimum and maximum pixel values and the "is signed" flag.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_min_max_64(
        const HIGEAR hIGear,
        const LPAT_RECT lpRect,
        LPAT_INT64 lpMin,
        LPAT_INT64 lpMax,
        LPBOOL lpSigned
);
```

Arguments:

Name	Type	Description
hIGear	const HIGEAR	The HIGEAR handle of the image to scan.
lpRect	const AT_RECT	Use this AT_RECT structure to specify the rectangular portion of the image to scan. Please see the ImageGear User's Manual if you are unfamiliar with this structure.
lpMin	LPAT_INT64	A far pointer to a AT_INT64which returns the value of the minimum pixel value in the region that was "scanned".
lpMax	LPAT_INT64	A far pointer to a AT_INT64 which returns the value of the maximum pixel value in the region that was "scanned".
lpSigned	LPBOOL	A far pointer to a BOOL which returns the status of the image sign. TRUE means the image is signed.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…32 bpp.

Example:

```
HIGEAR hIGear;
AT_ERRCOUNT nErrcount;
AT_RECT rcROI;
AT_INT64 lMin, lMax;
BOOL bSigned;
nErrcount = MED_IP_min_max_64(hIGear, &rcROI, &lMin, &lMax, &bSigned);
```

Remarks:

If the image is signed, then the returned values are also signed. It is OK to use this function for 8-16 bit images too, but in 32 bit operation systems it can bring to insignificant slowdown of performance.

Note that this function returns the min and max raw pixel values. That is, the returned values are not corrected using the Modality LUT (Rescale Slope/Intercept) values. To apply this correction to the min and max values that you get from MED_IP_min_max_64() call the function MED_DCM_DS_Rescale_get() which will return you the values of Rescale Slope and Rescale Intercept. Then use the following formulas:

min_corrected = (min_raw * rescale_slope) + rescale_intercept;

max_corrected = (max_raw * rescale_slope) + rescale_intercept;

Some modalities of DICOM use signed images. So if you know whether the image is signed or unsigned will help you to interpret the minimum and maximum values.

1.3.2.1.4.8 MED_IP_normalize

This function's main purpose is to convert the pixel data of a 16-bit image from signed to unsigned.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_normalize(
        HIGEAR hIGear,
        const LONG lMin
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	The HIGEAR handle to the image to normalize.
lMin	const LONG	A variable of type LONG used to set the minimum pixel value for the resulting image.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…32 bpp.

Remarks:

If you are using this function to convert the pixel data of a 16-bit image from signed to unsigned, set lMin to 0.

It can also be used to convert the minimum pixel value for the image. To do this, set lMin to greater than 0. This function searches the image for the minimum pixel value. It maps this value to lMin. Then, all pixel values are linearly adjusted to maintain the original contrast. If lMin is equal to the min pixel value in hIGear, then no change is made. The resulting image remains a 16-bit.

1.3.2.1.4.9 MED_IP_promote_to_16_gray

This function takes an 8-bit grayscale or color image and converts it to a grayscale image with a bit depth of 16 bpp.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_promote_to_16_gray(
        HIGEAR hIGear,
        const UINT iBits,
        const UINT iHighBit
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	The HIGEAR handle of the image to be converted.
iBits	const UINT	An integer variable that sets the number of bits that are actually used. The value range is 9 - 16 bits.
iHighBit	const UINT	An integer variable that sets the high bit for the new 16-bit image. The eighth bit of the original 8 bits is positioned here. If the image is saved as a DICOM, this information is saved to the High Bit Data Element (0028,0102).

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8 bpp.

Remarks:

The original 8-bit image is discarded. All images created by this function have 16-bit pixels, iBits sets the number of bits out of the 16 that are actually used. iHighBit sets the new position among the 16 bits at which the original 8 bits should be positioned.

1.3.2.1.4.10 MED_IP_reduce_depth_with_downshift

This function is used to downshift a chosen range of 8 bits (out of a maximum of 16) to the 256 pixel values that can be shown on an 8-bit display device.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_reduce_depth_with_downshift (
        HIGEAR hIGear,
        const UINT downshift
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle to the image.
downshift	const UINT	A UINT variable specifying the downshift value.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 9…16 bpp.

Remarks:

This function is used to downshift a chosen range of 8 bits (out of a maximum of 16) to the 256 pixel values that can be shown on an 8-bit display device. For example, if you set downshift to 8, and the image has 16 bpp, bits 8-15 will be downshifted and used alone as the pixel values.

This function will directly alter the pixel data.

See also MED_IP_reduce_depth_with_LUT().

1.3.2.1.4.11 MED_IP_reduce_depth_with_LUT

This function takes a 16-bit grayscale image and reduces it to a 8-bit grayscale one using provided (filled) LUT or current 16x8 display LUT.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_reduce_depth_with_LUT(
        HIGEAR hIGear,
        const LPBYTE lpLUT,
        const DWORD dwEntries
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	The HIGEAR handle to the image to convert.
lpLUT	const LPBYTE	A far pointer to the look-up table to use for reduction. Set to NULL if you want to use the display LUT.
dwEntries	const DWORD	A variable of type DWORD that specifies the number of entries in the LUT. This value is ignored if the display LUT is used.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 9…16 bpp.

Remarks:

The 16-bit image is discarded and replaced with the new 8-bit grayscale image.

To reduce memory requirements the LUT does not have to have a 16-bit input. dwEntries should hold the number of entries in the LUT. If there is a pixel found that can overflow the LUT it is ignored and replaced with a 0.

This function does not reduce 8-bit images. To reduce an 8-bit image, use the appropriate IG_IP_color_reduce_...() function from the baseline ImageGear API.

Once this function is called, the display LUT will be thrown away, since it is now an 8-bit image.

See also MED_IP_reduce_depth_with_downshift().

1.3.2.1.4.12 MED_IP_swap_bytes

This function swaps the 2 bytes of each 16-bit pixel in the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_IP_swap_bytes(HIGEAR hIGear);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	HIGEAR handle of the image.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 9…16 bpp.

Remarks:

This function can be used to help fix poorly constructed images which have the high and low bytes of each 16-bit grayscale pixel reversed. All pixels in the image are transformed.

1.3.2.1.5 Modality Transform Functions

This section provides information about the Modality Transform group of functions.

- MED_modality_transform_apply
- MED_modality_transform_apply_64

1.3.2.1.5.1 MED_modality_transform_apply

If hModalityLUT is not NULL, this function applies Modality LUT to the specified value.

Declaration:

```
AT_INT MED_modality_transform_apply(
        const AT_MED_MODALITY_RESCALE* lpRescale,
        HIGLUT hModalityLUT,
        AT_INT Value
);
```

Arguments:

Name	Type	Description
lpRescale	const AT_MED_MODALITY_RESCALE*	AT_MED_MODALITY_RESCALE structure.
hModalityLUT	HIGLUT	Modality LUT handle.
Value	AT_INT	Value to which to apply Modality transform.

Return Value:

Returns the resulting pixel value.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If hModalityLUT is NULL, this function applies the linear modality transform (lpRescale) to the specified value. For 17-32 bit values, please use MED_modality_transform_apply_64().

1.3.2.1.5.2 MED_modality_transform_apply_64

Applies the linear modality transform (lpRescale) to the specified value.

Declaration:

```
AT_INT MED_modality_transform_apply_64(
        const AT_MED_MODALITY_RESCALE* lpRescale,
        AT_INT64 Value,
        LPAT_INT64 lpResult
);
```

Arguments:

lpRescale	const AT_MED_MODALITY_RESCALE*	Pointer to AT_MED_MODALITY_RESCALE structure.
Value	AT_INT64	64-bit value to which to apply Modality transform.
lpResult	LPAT_INT64	Far pointer to 64 bit integer, where the function will put resulting value.

Return Value:

Returns the resulting pixel value.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.2.1.6 Presentation State Functions

This section provides information about the Presentation State group of functions.

- MED_PS_apply
- MED_PS_display_contrast
- MED_PS_display_contrast_auto
- MED_PS_display_contrast_auto_64
- MED_PS_extract
- MED_PS_GSDF_LUT_build
- MED_PS_GSDF_LUT_init
- MED_PS_pres_LUT_get
- MED_PS_pres_LUT_info_get
- MED_PS_pres_LUT_set
- MED_PS_pres_state_GSDF_apply

1.3.2.1.6.1 MED_PS_apply

This function extracts PS data from DICOM DataSet of hIGearPresState, and adjusts display of hIGear based on this data.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_apply (
        HIGEAR hIGear,
        HIGEAR hIGearPresState,
        DWORD dwFeatureFlags,
        DWORD dwGrpID,
        LPAT_MED_LUT_DESC lpGSDFLUT
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image to which Pres State shall be applied.
hIGearPresState	HIGEAR	Image containing Presentation State DataSet.
dwFeatureFlags	DWORD	Tells which features of PS to use - bit mask.
dwGrpID	DWORD	Display Group ID.
lpGSDFLUT	LPAT_MED_LUT_DESC	GSDF LUT info.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

hIGearPresState must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This function may result in the following:

- Adjust Display LUT associated with the image, using
  - VOI and Modality LUTs found in the presentation state DataSet
  - Presentation LUT found in the presentation state DataSet
  - GSDF LUT parameter
- Change display layout, including image rectangle, zoom and orientation
- Add ART objects

If there are no VOI or Modality LUT in the hIGearPresState, the function tries to find them in the DataSet of hIGear. If there are no such LUTs in hIGear, default values are taken.

If MED_PS_FEATURE_PRES_LUT option is specified, the function assumes that MED_PS_FEATURE_CONTRAST is selected as well.

1.3.2.1.6.2 MED_PS_display_contrast

Builds the 16x8 display LUT, using rescale, window, Presentation LUT and GSDF LUT.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_display_contrast (
        HIGEAR hIGear,
        DOUBLE rescale_slope,
        DOUBLE rescale_intercept,
        LONG window_center,
        LONG window_width,
        DOUBLE gamma,
        LPAT_MED_LUT_DESC lpPresLUTInfo,
        LPAT_MED_LUT_DESC lpGSDFLUTInfo,
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	16g image to have its LUT16x8 updated.
rescale_slope	DOUBLE	Rescale Slope (0028,1053).
rescale_intercept	DOUBLE	Rescale Intercept (0028,1054).
window_center	LONG	Window Center (0028,1050).
window_width	LONG	Window Width (0028,1051).
gamma	DOUBLE	Gamma correction - set to 1.0 to turn off.
lpPresLUTInfo	LPAT_MED_LUT_DESC	Presentation LUT info.
lpGSDFLUTInfo	LPAT_MED_LUT_DESC	GSDF LUT info.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…32 bpp.

Remarks:

If lpGSDFLUTInfo parameter is not NULL, gamma is not used.

1.3.2.1.6.3 MED_PS_display_contrast_auto

Builds the 16x8 display LUT, using rescale, Presentation LUT and GSDF LUT.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_display_contrast_auto (
        HIGEAR hIGear,
        LPAT_RECT lpRect,
        DOUBLE rescale_slope,
        DOUBLE rescale_intercept,
        DOUBLE gamma,
        LPAT_MED_LUT_DESC lpPresLUTInfo,
        LPAT_MED_LUT_DESC lpGSDFLUTInfo,
        LPLONG lpWindow_center,
        LPLONG lpWindow_width
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	16g image to have its LUT16x8 updated.
lpRect	LPAT_RECT	set to NULL to scan the entire image.
rescale_slope	DOUBLE	Rescale Slope (0028,1053).
rescale_intercept	DOUBLE	Rescale Intercept (0028,1054).
gamma	DOUBLE	Gamma correction - set to 1.0 to turn off.
lpPresLUTInfo	LPAT_MED_LUT_DESC	Presentation LUT info.
lpGSDFLUTInfo	LPAT_MED_LUT_DESC	GSDF LUT info.
lpWindow_center	LPLONG	Window Center (0028,1050).
lpWindow_width	LPLONG	Window Width (0028,1051).

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…16 bpp.

Remarks:

Window center and width are calculated automatically, based on min and max values of the image. If lpGSDFLUTInfo parameter is not NULL, gamma is not used.

For 17-32 bits per pixel images, please use MED_PS_display_contrast_auto_64().

1.3.2.1.6.4 MED_PS_display_contrast_auto_64

Builds the 16x8 display LUT, using rescale, Presentation LUT and GSDF LUT.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_display_contrast_auto_64 (
        HIGEAR hIGear,
        LPAT_RECT lpRect,
        DOUBLE rescale_slope,
        DOUBLE rescale_intercept,
        DOUBLE gamma,
        LPAT_INT64 lpWindow_center,
        LPAT_INT64 lpWindow_width
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	16g image to have its LUT16x8 updated.
lpRect	AT_RECT	set to NULL to scan the entire image.
rescale_slope	DOUBLE	Rescale Slope (0028,1053).
rescale_intercept	DOUBLE	Rescale Intercept (0028,1054).
gamma	DOUBLE	Gamma correction - set to 1.0 to turn off.
lpWindow_center	LPAT_INT64	Window Center (0028,1050) as 64 bit integer.
lpWindow_width	LPAT_INT64	Window Width (0028,1051) as 64 bit integer.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

Grayscale – 8…32 bpp.

Remarks:

Window center and width are calculated automatically, based on min and max values of the image. If lpGSDFLUTInfo parameter is not NULL, gamma is not used. Use this function for 17-32 bit grayscale images. Although you can also use this function for 8-16 bit images, performance may be affected in 32 bit operation systems.

1.3.2.1.6.5 MED_PS_extract

This function adds Presentation State tags to the DataSet of hIGearPresState, based on the display settings of hIGear, an ART marks attached to it.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_extract (
        HIGEAR hIGear,
        HIGEAR hIGearPresState,
        DWORD dwFeatureFlags,
        DWORD dwGrpID
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image whose settings should be exported to PS.
hIGearPresState	HIGEAR	Image that will contain Presentation State DataSet.
dwFeatureFlags	DWORD	Tells which features of PS to export - bit mask.
dwGrpID	DWORD	Display Group ID.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

Remarks:

If any of the tags are already present in the hIGearPresState, they are overwritten.

This function may add/update the following tags/sequences in the hIGearPresState:

- DCM_TAG_DisplayedAreaSelectionSequence
- DCM_TAG_GraphicAnnotationSequence
- DCM_TAG_GraphicLayerSequence
- DCM_TAG_ImageHorizontalFlip
- DCM_TAG_ImageRotation

To create a new presentation state DataSet, use the following steps:

1. Call IG_image_create_DIB_ex() with lCompression set to IG_BI_EMPTY.
2. Call MED_PS_extract() to add Presentation State tags.
3. Add the other tags required by the Presentation State module using general DataSet access functions.
4. Save Presentation State to file using IG_fltr_save_file() or similar functions.

To modify an existing Presentation State DataSet, use the following steps:

1. Load a Presentation State DataSet using IG_fltr_load_file() or similar functions.
2. Apply the Presentation State to a DICOM image, using MED_PS_apply().
3. Change display settings or ART marks.
4. Extract Presentation Data from DICOM image back to Presentation State DataSet.
5. Save Presentation State to file using IG_fltr_save_file() or similar functions.

Use MED_PS_pres_LUT_set() to add Presentation LUT to the Presentation State HIGEAR.

Use general DataSet functions to add VOI and Modality LUT to the Presentation State HIGEAR.

Use ImageGear image saving functions (nFormat=IG_FORMAT_DCM) to save lphIGearPresState into a presentation state file (.pre).

1.3.2.1.6.6 MED_PS_GSDF_LUT_build

This function builds a look-up table that maps pixel intensities from DICOM GSDF-compliant color space into pixel intensities of the display device.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_GSDF_LUT_build(
        DWORD dwCharactCurveEntryCount,
        LPDOUBLE lpCharactCurve,
        LPAT_MED_LUT_DESC lpLUT
);
```

Arguments:

Name	Type	Description
dwCharactCurveEntryCount	DWORD	Number of entries in Characteristic Curve.
lpCharactCurve	LPDOUBLE	Characteristic Curve of the display.
lpLUT	LPAT_MED_LUT_DESC	GSDF LUT to fill.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The user should fill in the members of the lpLUT structure and allocate the lpLUTData buffer. The space necessary for holding the LUT can be calculated using the AM_MED_LUT_SIZE_GET macro.

The LUT obtained from this function can be used for building the 16x8 or 8x8 display LUT. See MED_PS_apply(), MED_PS_display_contrast().

At the moment, ImageGear does not support any display devices that are capable of displaying more than 256 shades of grayscale. Hence, MED_PS_GSDF_LUT_build can only build a LUT consisting of 8-bit entries.

1.3.2.1.6.7 MED_PS_GSDF_LUT_init

This function initializes the HIGLUT object with GSDF LUT data.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_GSDF_LUT_init(
        DWORD dwCharactCurveEntryCount,
        LPDOUBLE lpCharactCurve,
        HIGLUT lut
);
```

Arguments:

Name	Type	Description
dwCharactCurveEntryCount	DWORD	Number of entries in Characteristic Curve.
lpCharactCurve	LPDOUBLE	Characteristic Curve of the display.
lut	HIGLUT	GSDF LUT to fill.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function replaces MED_PS_GSDF_LUT_build().

1.3.2.1.6.8 MED_PS_pres_LUT_get

This function fills the Presentation LUT with the data contained in the Presentation State DataSet.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_pres_LUT_get(
        HIGEAR hIGearPresState,
        LPAT_MED_LUT_DESC lpPresLUTInfo
);
```

Arguments:

Name	Type	Description
hIGearPresState	HIGEAR	Image that contains Presentation State DataSet .
lpPresLUTInfo	LPAT_MED_LUT_DESC	Presentation LUT info.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

1.3.2.1.6.9 MED_PS_pres_LUT_info_get

This function returns information about Presentation LUT contained in a Presentation State DataSet.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_pres_LUT_info_get(
        HIGEAR hIGearPresState,
        LPAT_MED_LUT_DESC lpPresLUTInfo
);
```

Arguments:

Name	Type	Description
hIGearPresState	HIGEAR	Image that contains Presentation State DataSet.
lpPresLUTInfo	LPAT_MED_LUT_DESC	Presentation LUT info.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

1.3.2.1.6.10 MED_PS_pres_LUT_set

This function adds Presentation LUT sequence to the DataSet of hIGearPresState.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_pres_LUT_set(
        HIGEAR hIGearPresState,
        LPAT_MED_LUT_DESC lpPresLUTInfo
);
```

Arguments:

Name	Type	Description
hIGearPresState	HIGEAR	Image that contains Presentation State DataSet.
lpPresLUTInfo	LPAT_MED_LUT_DESC	Presentation LUT info.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

Use this function together with MED_PS_extract().

1.3.2.1.6.11 MED_PS_pres_state_GSDF_apply

This function extracts PS data from DICOM DataSet of hIGearPresState, and adjusts display of hIGear based on this data.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_PS_pres_state_GSDF_apply(
        HIGEAR hIGear,
        HIGEAR hIGearPresState,
        DWORD dwFeatureFlags,
        DWORD dwGrpID,
        HIGLUT GSDFLUT
);
```

Arguments:

Name	Type	Description
hIGear	HIGEAR	Image to which Pres State shall be applied.
hIGearPresState	HIGEAR	Image containing Presentation State DataSet.
dwFeatureFlags	DWORD	Tells which features of PS to use - bit mask.
dwGrpID	DWORD	Display Group ID.
GSDFLUT	HIGLUT	GSDF LUT.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

All pixel formats supported by ImageGear Professional.

hIGearPresState must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Remarks:

This function may result in the following:

- Adjust Display LUT associated with the image, using
  - VOI and Modality LUTs found in the presentation state DataSet
  - Presentation LUT found in the presentation state DataSet
  - GSDF LUT parameter
- Change display layout, including image rectangle, zoom and orientation
- Add ART objects

If there are no VOI or Modality LUT in hIGearPresState, the function tries to find them in the DataSet of hIGear. If there are no such LUTs in hIGear, default values are taken.

If MED_PS_FEATURE_PRES_LUT option is specified, the function assumes that MED_PS_FEATURE_CONTRAST is selected.

This function replaces MED_PS_apply(). It provides the same functionality as MED_PS_apply(), but uses a more general type HIGLUT for GSDFLUT parameter.

1.3.2.1.7 Utility Functions

This section provides information about the Utility group of functions.

- MED_DCM_util_data_to_string
- MED_DCM_util_tag_info_add
- MED_DCM_util_tag_info_free
- MED_DCM_util_tag_info_get
- MED_DCM_util_VR_info_mode
- MED_DCM_util_VR_info_string

1.3.2.1.7.1 MED_DCM_util_data_to_string

This function takes the DICOM Data Field of the Current Data Element, designated in lpData, and converts it to a NULL-terminated character string.

Declaration:

```
BOOL ACCUAPI MED_DCM_util_data_to_string(
        const LPCHAR lpData,
        const AT_DCM_VR vr,
        const AT_DCM_VL vl,
        const INT first_item,
        const INT last_item,
        LPCHAR lpString,
        const DWORD string_len,
        const CHAR separator
);
```

Arguments:

Name	Type	Description
lpData	const LPCHAR	A far pointer to the data from a Data Element which you would like to convert to a string.
vr	const AT_DCM_VR	Set to the Value Representation (VR) for the Data. See enumIGMedVR for possible VR values.
vl	const AT_DCM_VL	Set to the Value Length of the Data.
first_item	const INT	First data value to process. Set to -1 to process all values.
last_item	const INT	Data value to stop processing at. This argument is only effective if first_item is not set to -1. If first_item is 0, set this argument to the number of values you'd like to process.
lpString	LPCHAR	A far pointer to a NULL-terminated string which returns the representation of the DICOM data field specified in lpData.
string_len	const DWORD	Set this to the length in bytes of lpString. If the length is shorter than the data being retrieved, it will simply truncate the data.
separator	const CHAR	Set this to the character that you would like to use to separate multiple data values; for example, you might set it to a comma or semicolon.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Set the vr argument to determine how the lpData is to be interpreted.

lpString can only be filled up to string_len - 1. 1 is subtracted from the string_len to accommodate the NULL. Any remainder is simply clipped.

For Data Fields with more than one data value, the separator character specified in separator is used to delimit the items.

1.3.2.1.7.2 MED_DCM_util_tag_info_add

This function allows you to add new entries into an internal table of Tag entries.

Declaration:

```
BOOL ACCUAPI MED_DCM_util_tag_info_add(
        const AT_DCM_TAG Tag,
        const AT_DCM_VR VR,
        const AT_DCM_VM VM,
        const WORD wVersion,
        const LPCHAR lpszTagName
);
```

Arguments:

Name	Type	Description
Tag	const AT_DCM_TAG	Set to a Tag value. The Tag must be supplied as a 32-bit value in which the first 16 bits (WORD) represent the Group Number and the second 16 bits represent the Element Number. Group and Element are expressed as WORDs.
lpVR	const AT_DCM_VR	Set to the VR of the new Tag. See enumIGMedVR for possible VR values.
lpVM	const AT_DCM_VM	Set to the VM of the new Tag. Value Multiplicity tells whether and/or how many items can be stored in this type of Data Element. See Remarks below.
wVersion	const WORD	Set this to the DICOM version. This should identify the first version of DICOM that includes this Tag. Most applications should set this to 3 but any value is accepted.
LpszTagName	const LPCHAR	Set to a character string name that should be provided as the description of this Tag.

Return Value:

Returns TRUE if the new Tag was successfully added to the Data Dictionary; FALSE otherwise.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

It can be used for adding newly defined DICOM Tags (new to the specification) or for adding private user-defined Tags. Once a new entry is added, the new Tag works just like all other Tags do.

- Set VM to a non-zero positive integer for a Tag which must contain a specific number of Items.
- Set VM to 0 for a Tag which can have an unlimited number of Items.
- Set VM to a negative integer for a Tag which can have a limited number of items up to the absolute value of the provided VM. For example, VM = -3 means that the Tag may have up to 3 items.

1.3.2.1.7.3 MED_DCM_util_tag_info_free

This function frees up the user-defined Data Dictionary, if it exists.

Declaration:

```
BOOL ACCUAPI MED_DCM_util_tag_info_free(VOID);
```

Arguments:

None

Remarks:

All User-Defined entries are discarded. The pre-defined Data Dictionary is not affected. This function returns TRUE if a table was freed. FALSE if there was no table to free.

Supported Raster Image Formats:

This function does not process image pixels.

Return Value:

TRUE if a table was freed; FALSE if there was no table to free.

1.3.2.1.7.4 MED_DCM_util_tag_info_get

This function returns information about the specified Tag.

Declaration:

```
BOOL ACCUAPI MED_DCM_util_tag_info_get(
        const AT_DCM_TAG Tag,
        LPAT_DCM_VR lpVR,
        LPAT_DCM_VM lpVM,
        LPWORD lpwVersion,
        LPCHAR lpszTagName
);
```

Arguments:

Name	Type	Description
Tag	const AT_DCM_TAG	Set to a Tag value. The Tag must be supplied as a 32-bit value in which the first 16 bits (WORD) represent the Group Number and the second 16 bits represent the Element Number.
lpVR	LPAT_DCM_VR	A far pointer which returns the current VR (Value Representation). Set to NULL if you don't need this information. See enumIGMedVR for possible VR values.
lpVM	LPAT_DCM_VM	A far pointer which returns the VM (Value Multiplicity) of the current Tag; set to NULL if you don't need this information. Value Multiplicity tells you whether and/or how many items can be stored in this type of Data Element. See Remarks below.
lpwVersion	LPWORD	Returns the version of DICOM Specification, such as 3.0.
lpszTagName	LPCHAR	Returns the name of the specified Tag.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

It returns the Value Representation, Value Multiplicity, DICOM version, and the Tag name. The function searches in the user-defined Data Dictionary first, and then if the Tag is not found, searches in the static Data Dictionary, which represents the Data Dictionary listed in Part 6 of the DICOM standard.

You might use this function before making a call that alters a Data Element.

- If lpVM returns a Positive integer: there must be this number of Items.
- If it returns a 0: you may have an unlimited number of items, including 0.
- If it returns a Negative integer: The number of items may include up to the absolute value of the value returned. For example, if lpVM = -3, you may have up to 3 items.

1.3.2.1.7.5 MED_DCM_util_VR_info_mode

This function looks up the Value Representation specified in vr_mode and returns the following information about it: its text representation, length, restrictions, and whether or not it can be a NULL-terminated string.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_util_VR_info_mode(
        const AT_DCM_VR vr_mode,
        LPCHAR lpVRstring,
        LPWORD lpwLength,
        LPWORD lpwRestriction,
        LPBOOL lpCheck_form,
        LPBOOL lpIsString
);
```

Arguments:

Name	Type	Description
vr_mode	const AT_DCM_VR	Set this to the type of VR (or data type) you would like information on. See enumIGMedVR for possible VR values.
lpVRstring	LPCHAR	A far pointer that returns the text representation of VR in 3 characters.
lpwLength	LPWORD	A far pointer that returns the size of the VR.
lpwRestriction	LPWORD	A far pointer that returns any restriction flags. These will be returned as constants that are defined in enumIGMedVRRestriction and begin with MED_DCM_LEN_.
lpReserved	LPBOOL	This argument has not been implemented yet. Please set to NULL for now.
lpIsString	LPBOOL	A far pointer to a BOOL value that tells you whether the data of this type of VR is a NULL-terminated string or not. If TRUE, data of this type is a NULL-terminated string and could be printed using the print format %s. If FALSE, the data is an integer or other binary data type.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.2.1.7.6 MED_DCM_util_VR_info_string

This function returns the type of Value Representation used.

Declaration:

```
AT_ERRCOUNT ACCUAPI MED_DCM_util_VR_info_string(
        const LPCHAR lpVR_string,
        LPWORD lpwLength,
        LPWORD lpwRestriction,
        LPBOOL lpReserved,
        LPBOOL lpIsString,
        LPAT_DCM_VR lpVr
);
```

Arguments:

Name	Type	Description
lpVRstring	const LPCHAR	Set to the 3-character text representation of the VR as used in the DICOM Specification, e.g., "PN", "CS", or "SQ." While these are 2-character codes, the end of line string termination makes it 3.
lpwLength	LPWORD	A far pointer that returns the size of the VR. This is the length in byes of a single instance of data that is of the specified type. For example, a VR of "SL" has a fixed length of 4 bytes per item. If the VM allows more than a single data value, then each one will take up 4 bytes. Other VRs have a maximum length. "PN" and "UI" both have a maximum of 64 bytes.
lpwRestriction	LPWORD	A far pointer that returns any restriction flags . These will be returned as constants that are defined in enumIGMedVRRestriction and begin with MED_DCM_LEN_.
lpReserved	LPBOOL	This argument has not been implemented yet. Please set to NULL for now.
lpIsString	LPBOOL	A far pointer to a BOOL value that tells you whether the data of this type of VR is a NULL-terminated string or not. If TRUE, data of this type is a NULL-terminated string and could be printed using the print format %s. If FALSE, the data is an integer or other binary data type.
lpVR	LPAT_DCM_VR	A far pointer that returns an AT_MODE constant that identifies the type of Value Representation. See enumIGMedVR for possible VR values.

Return Value:

Returns the number of ImageGear errors that occurred during the function call.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Supply a 3-character string (two alphabetic characters plus the end-of-string terminator), and this function will return one of the MED_DCM_VR_ constants defined in enumIGMedVR.

1.3.2.2 MD Component Macros Reference

This section provides information about the MD Component Macros, arranged in alphabetical order.

1.3.2.2.1 MED_DCM_DS_TAG_ELEMENT

This macro is used for getting the Element number out of a Tag.

Declaration:

```
MED_DCM_DS_TAG_ELEMENT(Tag);
```

Arguments:

Tag	Supply this argument with a 32-bit Tag value that begins with DCM_TAG_

Return Value:

Returns the 16-bit Element Number portion of a 32-bit Tag value.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
element = MED_DCM_DS_TAG_ELEMENT(DCM_TAG_IssuerOfPatientID)
/* element equals 0x0021                              */
```

Remarks:

Supply it with a 32-bit Tag value, and it will return the 16-bit Element Number value. The Element Number is the least significant WORD of the 32-bit Tag value.

1.3.2.2.2 MED_DCM_DS_TAG_GROUP

This macro is used for getting the Group Number out of a Tag.

Declaration:

```
MED_DCM_DS_TAG_GROUP(Tag)
```

Arguments:

Tag	Supply this argument with a 32-bit Tag value.

Return Value:

Returns the 16-bit Group Number portion of a 32-bit Tag value.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
group = MED_DCM_DS_TAG_GROUP(DCM_TAG_IssuerOfPatientID)
/* group equals 0x0010                               */
```

Remarks:

Supply it with a 32-bit Tag value, and it will return the 16-bit Group Number value. The Group Number is the most significant WORD of the 32-bit Tag value.

1.3.2.2.3 MED_DCM_DS_TAG_MAKE

This macro takes a 16-bit Group Number and a 16-bit Element Number and returns a 32-bit Tag value.

Declaration:

```
MED_DCM_DS_TAG_MAKE(gn, en);
```

Arguments:

gn	Supply this with a valid DICOM Group Number.
en	Supply this with a valid DICOM Element Number.

Return Value:

Returns the 32-bit Tag value from the 16-bit Group Number and 16-bit Element Number that you provide.

Supported Raster Image Formats:

This function does not process image pixels.

The image must have a DICOM DataSet attached to it. Use MED_DCM_DS_exists to check whether the image contains a DataSet.

Example:

```
AT_DCM_TAG Tag; Tag = MED_DCM_DS_TAG_MAKE(0x0010, 0x0021) /* Tag equals
DCM_TAG_IssuerOfPatientID or 0x00100021 */
```

Remarks:

The returned Tag is compatible with the enumIGMedTag enumeration.

1.3.2.3 MD Component Structures Reference

This section provides information about the MD Component Structures, arranged in alphabetical order.

- AT_MED_DCM_DISPLAY_SETTINGS
- AT_MED_MODALITY_RESCALE
- AT_MED_PIXEL_PADDING_SETTINGS
- AT_MED_VOI_WINDOW

1.3.2.3.1 AT_MED_DCM_DISPLAY_SETTINGS

This structure contains settings that can be used for displaying DICOM images with proper contrast.

Declaration:

```
struct AT_MED_DCM_DISPLAY_SETTINGS
{
        AT_MED_MODALITY_RESCALE ModalityRescale;
        HIGLUT ModalityLUT;
        AT_MED_VOI_WINDOW VOIWindow;
        HIGLUT VOILUT;
        AT_BOOL IsInverted;
        AT_DOUBLE Gamma;
        HIGLUT PresentationLUT;
        HIGLUT GSDFLUT;
        AT_MED_PIXEL_PADDING_SETTINGS PixelPadding;
};
typedef struct AT_MED_DCM_DISPLAY_SETTINGS;
```

Members:

Name	Type	Description
ModalityRescale	AT_MED_MODALITY_RESCALE	Specifies linear Modality transform (Rescale Slope/Intercept). Ignored if ModalityLUT is not NULL.
ModalityLUT	HIGLUT	Specifies Modality LUT.
VOIWindow	AT_MED_VOI_WINDOW	Specifies linear VOI transform (Window center/width). Ignored if VOI LUT is not NULL.
VOILUT	HIGLUT	Specifies VOI LUT.
IsInverted	AT_BOOL	Tells whether the image should be displayed inverted. TRUE corresponds to MONOCHROME1 photometric interpretation, FALSE corresponds to MONOCHROME2 photometric interpretation.
Gamma	AT_DOUBLE	Specifies gamma correction value. Ignored if GSDFLUT is not NULL.
PresentationLUT	HIGLUT	Specifies Presentation LUT.
GSDFLUT	HIGLUT	Specifies Grayscale Standard Display Function LUT.
PixelPadding	AT_MED_PIXEL_PADDING_SETTINGS	Specifies pixel padding settings.

Remarks:

The settings are applied in the following order: Modality (LUT or Rescale), VOI (LUT or Window), IsInverted flag, Presentation LUT, Gamma or GSDFLUT.

1.3.2.3.2 AT_MED_MODALITY_RESCALE

This structure specifies linear Modality transform.

Declaration:

```
struct AT_MED_MODALITY_RESCALE
{
        AT_DOUBLE Slope;
        AT_DOUBLE Intercept;
};
typedef struct AT_MED_MODALITY_RESCALE;
```

Members:

Name	Type	Description
Slope	AT_DOUBLE	Rescale slope.
Intercept	AT_DOUBLE	Rescale Intercept.

1.3.2.3.3 AT_MED_PIXEL_PADDING_SETTINGS

This structure represents DICOM pixel padding settings.

Declaration:

```
struct AT_MED_PIXEL_PADDING_SETTINGS
{
        AT_BOOL UsePixPadding;
        AT_INT PixPaddingValue;
        AT_INT ShowPaddingAs;
};
```

Members:

Name	**Type**	**Description**
UsePixPadding	AT_BOOL	Specifies whether pixel padding should be taken into account during display.
PixPaddingValue	AT_INT	Specifies pixel padding value.
ShowPaddingAs	AT_INT	Specifies the output intensity for displaying padding pixels.

1.3.2.3.4 AT_MED_VOI_WINDOW

This structure specifies linear VOI transform.

Declaration:

```
struct AT_MED_VOI_WINDOW
{
        AT_INT Center;
        AT_INT Width;
};
typedef struct AT_MED_VOI_WINDOW;
```

Members:

Name	Type	Description
Center	AT_INT	Window center.
Width	AT_INT	Window width.

1.3.2.4 MD Component Enumerations Reference

This section provides information about the MD Component Enumerations, arranged in alphabetical order.

- enumIGMedColorSchemes
- enumIGMedLevelOption
- enumIGMedPhotoInt
- enumIGMedPixelRep
- enumIGMedPlanarConfig
- enumIGMedPSFeatureFlags
- enumIGMedSOP
- enumIGMedTag
- enumIGMedTS
- enumIGMedVR
- enumIGMedVRRestriction

1.3.2.4.1 enumIGMedColorSchemes

Identifies medical pseudo-coloring schemes.

Values:

MED_PSEUDOCOLOR_OFF	Reset to Grayscale.
MED_PSEUDOCOLOR_OIL_FILM	Oil Film.
MED_PSEUDOCOLOR_DARK_BLUE_TO_BRIGHT_RED	Dark Blue to Bright Red.
MED_PSEUDOCOLOR_GREEN_TO_RED	Green to Red.
MED_PSEUDOCOLOR_RED_GREEN_BLUE	Red, Green, Blue.
MED_PSEUDOCOLOR_THERMAL	Thermal.
MED_PSEUDOCOLOR_BRIGHT_RAINBOW	Bright Rainbow.

1.3.2.4.2 enumIGMedLevelOption

Identifies DICOM Element List level navigation options.

Values:

MED_DCM_MOVE_LEVEL_FIXED	This setting tells the method to move only within the same level as the previous Current Data Element. Any SQs and all of their Data Elements will be skipped over. If you are in an SQ, you can only move within the SQ.
MED_DCM_MOVE_LEVEL_FLOAT	This setting tells the method to move up or down as needed to get to the next DE. If the next DE is an SQ the Current DE moves down into it. At the end of the SQ, the Current DE moves back to the lower levels (for example, from Level 2 to Level 1).

1.3.2.4.3 enumIGMedPhotoInt

Identifies DICOM Photometric Interpretations. See DICOM specification PS 3.3 C.7.6.3.1.2 for more details.

Values:

MED_DCM_PHOTO_INT_ARGB	This value has been deprecated and will be removed from the public API in a future release.
MED_DCM_PHOTO_INT_CMYK	This value has been deprecated and will be removed from the public API in a future release.
MED_DCM_PHOTO_INT_HSV	This value has been deprecated and will be removed from the public API in a future release.
MED_DCM_PHOTO_INT_MONOCHROME1	Pixel data represent a single monochrome image plane. The minimum sample value is intended to be displayed as white.
MED_DCM_PHOTO_INT_MONOCHROME2	Pixel data represent a single monochrome image plane. The minimum sample value is intended to be displayed as black.
MED_DCM_PHOTO_INT_PALETTE_COLOR	Pixel data describe a color image with a single sample per pixel (single image plane). The pixel value is used as an index into each of the Red, Blue, and Green Palette Color Lookup Tables
MED_DCM_PHOTO_INT_RGB	Pixel data represent a color image described by red, green, and blue image planes.
MED_DCM_PHOTO_INT_UNKNOWN	Photometric interpretation is unknown.
MED_DCM_PHOTO_INT_YBR_FULL	Pixel data represent a color image described by one luminance (Y) and two chrominance planes (Cb and Cr).
MED_DCM_PHOTO_INT_YBR_FULL_422	The same as YBR_FULL except that the Cb and Cr values are sampled horizontally at half the Y rate and as a result there are half as many Cb and Cr values as Y values.
MED_DCM_PHOTO_INT_YBR_ICT	Irreversible Color Transformation. Pixel data represent a color image described by one luminance (Y) and two chrominance planes (Cb and Cr). Used with JPEG 2000 transfer syntax.
MED_DCM_PHOTO_INT_YBR_PARTIAL_422	The same as YBR_FULL_422 with certain limitations. See DICOM specification PS 3.3 for more detail.
MED_DCM_PHOTO_INT_YBR_RCT	Reversible Color Transformation. Pixel data represent a color image described by one luminance (Y) and two chrominance planes (Cb and Cr). Used with JPEG 2000 transfer syntax.

1.3.2.4.4 enumIGMedPixelRep

Identifies DICOM Pixel Representations.

Values:

MED_DCM_PIXEL_REP_2S_COMPLEMENT	2's complement signed integer.
MED_DCM_PIXEL_REP_UNKNOWN	Unknown.
MED_DCM_PIXEL_REP_UNSIGNED	Unsigned integer.

1.3.2.4.5 enumIGMedPlanarConfig

Identifies DICOM Planar Configurations.

Values:

MED_DCM_PLANAR_PIXEL_BY_PIXEL	Pixels are stored pixel-by-pixel.
MED_DCM_PLANAR_PLANE_BY_PLANE	Pixels are stored plane-by-plane.

1.3.2.4.6 enumIGMedPSFeatureFlags

A bitmask that specifies features of the Presentation State object that can be applied or extracted separately.

Values:

MED_PS_FEATURE_NONE	No feature.
MED_PS_FEATURE_PRES_LUT	Presentation LUT.
MED_PS_FEATURE_VOI_LUT	VOI LUT.
MED_PS_FEATURE_MODALITY_LUT	Modality LUT.
MED_PS_FEATURE_DISPLAYED_AREA	Reserved for future use.
MED_PS_FEATURE_ORIENTATION	Orientation.
MED_PS_FEATURE_ANNOTAIONS	Annotations.
MED_PS_FEATURE_ALL	All features.

Remarks:

See MED_PS_apply and MED_PS_extract for more details.

1.3.2.4.7 enumIGMedSOP

Specifies DICOM Service Object Pair (SOP) constants.

Values:

MED_DCM_SOP_NULL	Null class.
MED_DCM_SOP_VERIFICATION	Verification SOP Class.
MED_DCM_SOP_MEDIA_STORAGE_DIR_STORAGE	Media Storage Directory Storage.
MED_DCM_SOP_HOT_IRON_COLOR_PALETTE_INSTANCE	Hot Iron Color Palette SOP Instance (Well-known).
MED_DCM_SOP_PET_COLOR_PALETTE_INSTANCE	PET Color Palette SOP Instance (Well-known).
MED_DCM_SOP_HOT_METAL_BLUE_COLOR_PALETTE_INSTANCE	Hot Metal Blue Color Palette SOP Instance (Well-known).
MED_DCM_SOP_PET_20_STEP_COLOR_PALETTE_INSTANCE	PET 20 Step Color Palette SOP Instance (Well-known).
MED_DCM_SOP_BASIC_STUDY_CONTENT_NOTIFICATION	Basic Study Content Notification SOP Class (Retired).
MED_DCM_SOP_STORAGE_COMMITMENT_PUSH_CLASS	Storage Commitment Push Model SOP Class.
MED_DCM_SOP_STORAGE_COMMITMENT_PUSH_INSTANCE	Storage Commitment Push Model SOP Instance (Well-known).
MED_DCM_SOP_STORAGE_COMMITMENT_PULL_CLASS	Storage Commitment Pull Model SOP Class (Retired).
MED_DCM_SOP_STORAGE_COMMITMENT_PULL_INSTANCE	Storage Commitment Pull Model SOP Instance (Well-known) (Retired).
MED_DCM_SOP_PROCEDURAL_EVENT_LOGGING	Procedural Event Logging SOP Class.
MED_DCM_SOP_PROCEDURAL_EVENT_LOGGING_INSTANCE	Procedural Event Logging SOP Instance (Well-known).
MED_DCM_SOP_SUBSTANCE_ADMINISTRATION_LOGGING	Substance Administration Logging SOP Class.
MED_DCM_SOP_SUBSTANCE_ADMINISTRATION_LOGGING_INSTANCE	Substance Administration Logging SOP Instance (Well-known).
MED_DCM_SOP_DICOM_UID_REGISTRY	DICOM UID Registry (DICOM UIDs as a Coding Scheme).
MED_DCM_SOP_DICOM_CONTROLLED_TERMINOLOGY	DICOM Controlled Terminology (Coding Scheme).
MED_DCM_SOP_DICOM_APP_CONTEXT	DICOM Application Context Name.
MED_DCM_SOP_PAT_MGMT_DET	Detached Patient Management SOP Class (Retired).
MED_DCM_SOP_PAT_MGMT_META	Detached Patient Management Meta SOP Class (Retired).
MED_DCM_SOP_VISIT_MGMT_DET	Detached Visit Management SOP Class (Retired).
MED_DCM_SOP_STUDY_MGMT_DET	Detached Study Management SOP Class (Retired).

MED_DCM_SOP_STUDY_MGMT_COMP	Study Component Management SOP Class (Retired).
MED_DCM_SOP_MOD_PERF_PROC_STEP	Modality Performed Procedure Step SOP Class.
MED_DCM_SOP_MOD_PERF_PROC_STEP_RETRIEVE	Modality Performed Procedure Step Retrieve SOP Class.
MED_DCM_SOP_MOD_PERF_PROC_STEP_NOTIFY	Modality Performed Procedure Step Notification SOP Class.
MED_DCM_SOP_RESULT_MGMT_DET	Detached Results Management SOP Class (Retired).
MED_DCM_SOP_RESULT_MGMT_META	Detached Results Management Meta SOP Class (Retired).
MED_DCM_SOP_STUDY_MGMT_META	Detached Study Management Meta SOP Class (Retired).
MED_DCM_SOP_INTERP_MGMT_DET	Detached Interpretation Management SOP Class (Retired).
MED_DCM_SOP_STORAGE_SERVICE_CLASS	Storage Service Class.
MED_DCM_SOP_BASIC_FILM_SESSION	Basic Film Session SOP Class.
MED_DCM_SOP_BASIC_FILM_BOX	Basic Film Box SOP Class.
MED_DCM_SOP_BASIC_GRAY_IMG_BOX	Basic Grayscale Image Box SOP Class.
MED_DCM_SOP_BASIC_COLOR_IMG_BOX	Basic Color Image Box SOP Class.
MED_DCM_SOP_REF_IMG_BOX	Reference Image Box SOP Class (Retired).
MED_DCM_SOP_BASIC_GRAY_PRINT_MGMT_META	Basic Grayscale Print Management Meta SOP Class.
MED_DCM_SOP_REF_GRAY_PRINT_MGMT_META	Referenced Grayscale Print Management Meta SOP Class (Retired).
MED_DCM_SOP_PRINT_JOB	Print Job SOP Class.
MED_DCM_SOP_BASIC_ANNOTATION_BOX	Basic Annotation Box SOP Class.
MED_DCM_SOP_PRINTER	Printer SOP Class.
MED_DCM_SOP_PRINTER_CONFIGURATION_RETRIEVAL	Printer Configuration Retrieval SOP Class.
MED_DCM_SOP_PRINTER_INSTANCE	Printer SOP Instance (Well-known).
MED_DCM_SOP_PRINTER_CONFIGURATION_RETRIEVAL_INSTANCE	Printer Configuration Retrieval SOP Instance (Well-known).
MED_DCM_SOP_BASIC_COLOR_PRINT_MGMT_META	Basic Color Print Management Meta SOP Class.
MED_DCM_SOP_REF_COLOR_PRINT_MGMT_META	Referenced Color Print Management Meta SOP class (Retired).
MED_DCM_SOP_VOI_LUT_BOX	VOI LUT Box SOP Class.
MED_DCM_SOP_PRESENTATION_LUT	Presentation LUT SOP Class.
MED_DCM_SOP_IMG_OVLY_BOX	Image Overlay Box SOP Class (Retired).
MED_DCM_SOP_BASIC_PRINT_IMAGE_OVERLAY_BOX	Basic Print Image Overlay Box SOP Class (Retired).

MED_DCM_SOP_PRINT_QUEUE_INSTANCE	Print Queue SOP Instance (Well-known) (Retired).
MED_DCM_SOP_PRINT_QUEUE_MGMT	Print Queue Management SOP Class (Retired).
MED_DCM_SOP_STORED_PRINT_STORAGE	Stored Print Storage SOP Class (Retired).
MED_DCM_SOP_HARDCOPY_GRAYSCALE_STORAGE	Hardcopy Grayscale Image Storage SOP Class (Retired).
MED_DCM_SOP_HARDCOPY_COLOR_STORAGE	Hardcopy Color Image Storage SOP Class (Retired).
MED_DCM_SOP_PULL_PRINT_REQUEST	Pull Print Request SOP Class (Retired).
MED_DCM_SOP_PULL_STORED_PRINT_MGMT_META	Pull Stored Print Management Meta SOP Class (Retired).
MED_DCM_SOP_MEDIA_CREATION_MANAGEMENT_UID	Media Creation Management UID SOP Class.
MED_DCM_SOP_CR_STORAGE	Computed Radiography Image Storage.
MED_DCM_SOP_DIGI_XRAY_PRES_IMG_STORAGE	Digital X-Ray Image Storage - For Presentation.
MED_DCM_SOP_DIGI_XRAY_PROC_IMG_STORAGE	Digital X-Ray Image Storage - For Processing.
MED_DCM_SOP_DIGI_MAMMO_PRES_IMG_STORAGE	Digital Mammography X-Ray Image Storage - For Presentation.
MED_DCM_SOP_DIGI_MAMMO_PROC_IMG_STORAGE	Digital Mammography X-Ray Image Storage - For Processing.
MED_DCM_SOP_DIGI_INTRA_ORAL_PRES_IMG_STORAGE	Digital Intra-oral X-Ray Image Storage - For Presentation.
MED_DCM_SOP_DIGI_INTRA_ORAL_PROC_IMG_STORAGE	Digital Intra-oral X-Ray Image Storage - For Processing.
MED_DCM_SOP_CT_STORAGE	CT Image storage.
MED_DCM_SOP_CT_ENHANCED_STORAGE	Enhanced CT Image Storage.
MED_DCM_SOP_USMF_STORAGE__RET	Ultrasound Multi-frame Image Storage (Retired).
MED_DCM_SOP_USMF_STORAGE	Ultrasound Multi-frame Image Storage.
MED_DCM_SOP_MR_STORAGE	MR Image Storage.
MED_DCM_SOP_MR_ENHANCED_IMAGE_STORAGE	Enhanced MR Image Storage.
MED_DCM_SOP_MR_SPECTROSCOPY_STORAGE	MR Spectroscopy Storage.
MED_DCM_SOP_ENHANCED_MR_COLOR_IMAGE_STORAGE	Enhanced MR Color Image Storage.
MED_DCM_SOP_NM_STORAGE__RET	Nuclear Medicine Image Storage (Retired).
MED_DCM_SOP_US_STORAGE__RET	Ultrasound Image Storage (Retired).
MED_DCM_SOP_US_STORAGE	Ultrasound Image Storage.
MED_DCM_SOP_ENHANCED_US_VOLUME_STORAGE	Enhanced US Volume Storage.
MED_DCM_SOP_SC_STORAGE	Secondary Capture Image Storage.
MED_DCM_SOP_SC_MF_SINGLE_BIT_IMAGE_STORAGE	Multiframe Single Bit Secondary

	Capture Image Storage.
MED_DCM_SOP_SC_MF_GRAYSCALE_BYTE_IMAGE_STORAGE	Multiframe Grayscale Byte Secondary Capture Image Storage.
MED_DCM_SOP_SC_MF_GRAYSCALE_WORD_IMAGE_STORAGE	Multiframe Grayscale Word Secondary Capture Image Storage.
MED_DCM_SOP_SC_MF_TRUE_COLOR_IMAGE_STORAGE	Multiframe True Color Secondary Capture Image Storage.
MED_DCM_SOP_OVERLAY_STORAGE	Standalone Overlay Storage (Retired).
MED_DCM_SOP_CURVE_STORAGE	Standalone Curve Storage (Retired).
MED_DCM_SOP_WAVEFORM_STORAGE	Waveform Storage - Trial (Retired).
MED_DCM_SOP_WAVEFORM_ECG_STORAGE	General ECG Waveform Storage.
MED_DCM_SOP_WAVEFORM_AUDIO_STORAGE	Waveform Audio Storage.
MED_DCM_SOP_AMBULATORY_ECG_WAVEFORM_STORAGE	Ambulatory ECG Waveform Storage.
MED_DCM_SOP_WAVEFORM_HEMO_STORAGE	Hemodynamic Waveform Storage.
MED_DCM_SOP_CARDIAC_ELECTROPHYSIOLOGY_WAVEFORM_STORAGE	Cardiac Electrophysiology Waveform Storage.
MED_DCM_SOP_BASIC_VOICE_AUDIO_WAVEFORM_STORAGE	Basic Voice Audio Waveform Storage.
MED_DCM_SOP_GENERAL_AUDIO_WAVEFORM_STORAGE	General Audio Waveform Storage.
MED_DCM_SOP_ARTERIAL_PULSE_WAVEFORM_STORAGE	Arterial Pulse Waveform Storage.
MED_DCM_SOP_RESPIRATORY_WAVEFORM_STORAGE	Respiratory Waveform Storage.
MED_DCM_SOP_MOD_LUT_STORAGE	Standalone Modality LUT Storage (Retired).
MED_DCM_SOP_VOI_LUT_STORAGE	Standalone VOI LUT Storage (Retired).
MED_DCM_SOP_GRAY_SOFTCOPY_PRES_STATE_STORAGE	Grayscale Softcopy Presentation State Storage SOP Class.
MED_DCM_SOP_COLOR_SOFTCOPY_PRES_STATE_STORAGE	Color Softcopy Presentation State Storage SOP Class.
MED_DCM_SOP_PSEUDO_COLOR_SOFTCOPY_PRES_STATE_STORAGE	Pseudo-Color Softcopy Presentation State Storage SOP Class.
MED_DCM_SOP_BLENDING_SOFTCOPY_PRES_STATE_STORAGE	Blending Softcopy Presentation State Storage SOP Class.
MED_DCM_SOP_XA_XRF_GRAYSCALE_SOFTCOPY_PRES_STATE_STORAGE	XA/XRF Grayscale Softcopy Presentation State Storage.
MED_DCM_SOP_XRAY_ANGIO_STORAGE	X-Ray Angiographic Image Storage.
MED_DCM_SOP_XRAY_ENHANCED_XA_IMAGE_STORAGE	Enhanced XA Image Storage.
MED_DCM_SOP_XRAY_RF_STORAGE	X-Ray Radiofluoroscopic Image Storage.

MED_DCM_SOP_ENHANCED_XRF_IMAGE_STORAGE	Enhanced XRF Image Storage.
MED_DCM_SOP_XRAY_ANGIO_BI_PLANE_STORAGE	X-Ray Angiographic Bi-Plane Image storage (Retired).
MED_DCM_SOP_XRAY_3D_ANGIOGRAPHIC_STORAGE	X-Ray 3D Angiographic Image Storage.
MED_DCM_SOP_XRAY_3D_CRANIOFACIAL_STORAGE	X-Ray 3D Craniofacial Image Storage.
MED_DCM_SOP_BREAST_TOMOSYNTHESIS_IMAGE_STORAGE	Breast Tomosynthesis Image Storage.
MED_DCM_SOP_NM_STORAGE	Nuclear Medicine Image storage.
MED_DCM_SOP_RAW_DATA_STORAGE	RAW data storage.
MED_DCM_SOP_SPATIAL_REGISTRATION_STORAGE	Spatial Registration Storage.
MED_DCM_SOP_SPATIAL_FIDUCIALS_STORAGE	Spatial Fiducials Storage.
MED_DCM_SOP_DEFORMABLE_SPATIAL_REGISTRATION_STORAGE	Deformable Spatial Registration Storage.
MED_DCM_SOP_SEGMENTATION_STORAGE	Segmentation Storage.
MED_DCM_SOP_SURFACE_SEGMENTATION_STORAGE	Surface Segmentation Storage.
MED_DCM_SOP_REAL_WORLD_VALUE_MAPPING_STORAGE	Real World Value Mapping Storage.
MED_DCM_SOP_VL_IMG_STORAGE	VL Image Storage - Trial (Retired).
MED_DCM_SOP_VL_MULTIFRAME_IMG_STORAGE	VL Multi-frame Image Storage - Trial (Retired).
MED_DCM_SOP_VL_ENDO_IMG_STORAGE	VL Endoscopic Image Storage.
MED_DCM_SOP_VIDEO_ENDOSCOPIC_IMAGE_STORAGE	Video Endoscopic Image Storage.
MED_DCM_SOP_VL_MICRO_IMG_STORAGE	VL Microscopic Image Storage.
MED_DCM_SOP_VIDEO_MICROSCOPIC_IMAGE_STORAGE	Video Microscopic Image Storage.
MED_DCM_SOP_VL_SLIDE_MICRO_IMG_STORAGE	VL Slide-Coordinates Microscopic Image Storage.
MED_DCM_SOP_VL_PHOTO_IMG_STORAGE	VL Photographic Image Storage.
MED_DCM_SOP_VIDEO_PHOTOGRAPHIC_IMAGE_STORAGE	Video Photographic Image Storage.
MED_DCM_SOP_OPHTHALMIC_PHOTOGRAPHY_8_BIT_IMAGE_STORAGE	Ophthalmic Photography 8 Bit Image Storage.
MED_DCM_SOP_OPHTHALMIC_PHOTOGRAPHY_16_BIT_IMAGE_STORAGE	Ophthalmic Photography 16 Bit Image Storage.
MED_DCM_SOP_STEREOMETRIC_RELATIONSHIP_STORAGE	Stereometric Relationship Storage.
MED_DCM_SOP_OPHTHALMIC_TOMOGRAPHY_IMAGE_STORAGE	Ophthalmic Tomography Image Storage.
MED_DCM_SOP_LENSOMETRY_MEASUREMENTS_STORAGE	Lensometry Measurements Storage.
MED_DCM_SOP_AUTOREFRACTION_MEASUREMENTS_STORAGE	Autorefraction Measurements Storage.
MED_DCM_SOP_KERATOMETRY_MEASUREMENTS_STORAGE	Keratometry Measurements Storage.
MED_DCM_SOP_SUBJECTIVE_REFRACTION_MEASUREMENTS_STORAGE	Subjective Refraction

	Measurements Storage.
MED_DCM_SOP_VISUAL_ACUITY_MEASUREMENTS_STORAGE	Visual Acuity Measurements Storage.
MED_DCM_SOP_SPECTACLE_PRESCRIPTION_REPORTS_STORAGE	Spectacle Prescription Reports Storage.
MED_DCM_SOP_MACULAR_GRID_THICKNESS_AND_VOLUME_REPORT_STORAGE	Macular Grid Thickness and Volume Report Storage.
MED_DCM_SOP_SR_TEXT_STORAGE	Text SR Storage - Trial (Retired).
MED_DCM_SOP_SR_AUDIO_STORAGE	Audio SR Storage - Trial (Retired).
MED_DCM_SOP_SR_DETAIL_STORAGE	Detail SR Storage - Trial (Retired).
MED_DCM_SOP_SR_COMPREHENSIVE_STORAGE	Comprehensive SR Storage - Trial (Retired).
MED_DCM_SOP_SR_BASIC_TEXT	Basic Text SR Storage.
MED_DCM_SOP_SR_ENHANCED	Enhanced SR Storage.
MED_DCM_SOP_SR_COMPREHENSIVE	Comprehensive SR Storage.
MED_DCM_SOP_PROCEDURE_LOG_STORAGE	Procedure Log Storage.
MED_DCM_SOP_SR_MAMMO_CAD	Mammography CAD SR Storage.
MED_DCM_SOP_KEY_OBJECT_SELECTION_DOCUMENT	Key Object Selection Document Storage.
MED_DCM_SOP_SR_CHEST_CAD	Chest CAD SR Storage.
MED_DCM_SOP_XRAY_RADIATION_DOSE_SR_STORAGE	X-Ray Radiation Dose SR Storage.
MED_DCM_SOP_COLON_CAD_SR_STORAGE	Colon CAD SR Storage.
MED_DCM_SOP_ENCAPSULATED_PDF_STORAGE	Encapsulated PDF Storage.
MED_DCM_SOP_ENCAPSULATED_CDA_STORAGE	Encapsulated CDA Storage.
MED_DCM_SOP_PET_STORAGE	Positron Emission Tomography Image Storage.
MED_DCM_SOP_PET_CURVE_STORAGE	Standalone PET Curve Storage (Retired).
MED_DCM_SOP_ENHANCED_PET_IMAGE_STORAGE	Enhanced PET Image Storage.
MED_DCM_SOP_BASIC_STRUCTURED_DISPLAY_STORAGE	Basic Structured Display Storage.
MED_DCM_SOP_RT_IMG_STORAGE	RT image storage.
MED_DCM_SOP_RT_DOSE_STORAGE	RT Dose Storage.
MED_DCM_SOP_RT_STRUCTURE_SET_STORAGE	RT Structure Set Storage.
MED_DCM_SOP_RT_TREATMENT_RECORD_STORAGE	RT Beams Treatment Record Storage.
MED_DCM_SOP_RT_PLAN_STORAGE	RT Plan Storage.
MED_DCM_SOP_RT_BRACHY_TREATMENT_RECORD_STORAGE	RT Brachy Treatment Record Storage.
MED_DCM_SOP_RT_TREATMENT_SUMMARY_RECORD_STORAGE	RT Treatment Summary Record Storage.
MED_DCM_SOP_RT_ION_PLAN_STORAGE	RT Ion Plan Storage.
MED_DCM_SOP_RT_ION_BEAMS_TREATMENT_RECORD_STORAGE	RT Ion Beams Treatment Record Storage.
MED_DCM_SOP_PAT_ROOT_QR_FIND	Patient Root Query/Retrieve

	Information Model - FIND.
MED_DCM_SOP_PAT_ROOT_QR_MOVE	Patient Root Query/Retrieve Information Model - MOVE.
MED_DCM_SOP_PAT_ROOT_QR_GET	Patient Root Query/Retrieve Information Model - GET.
MED_DCM_SOP_STUDY_ROOT_QR_FIND	Study Root Query/Retrieve Information Model - FIND.
MED_DCM_SOP_STUDY_ROOT_QR_MOVE	Study Root Query/Retrieve Information Model - MOVE.
MED_DCM_SOP_STUDY_ROOT_QR_GET	Study Root Query/Retrieve Information Model - GET.
MED_DCM_SOP_PAT_STUDY_ROOT_QR_FIND	Patient/Study Only Query/Retrieve Information Model - FIND (Retired).
MED_DCM_SOP_PAT_STUDY_ROOT_QR_MOVE	Patient/Study Only Query/Retrieve Information Model - MOVE (Retired).
MED_DCM_SOP_PAT_STUDY_ROOT_QR_GET	Patient/Study Only Query/Retrieve Information Model - GET (Retired).
MED_DCM_SOP_COMPOSITE_INSTANCE_ROOT_RETRIEVE_MOVE	Composite Instance Root Retrieve - MOVE.
MED_DCM_SOP_COMPOSITE_INSTANCE_ROOT_RETRIEVE_GET	Composite Instance Root Retrieve - GET.
MED_DCM_SOP_COMPOSITE_INSTANCE_RETRIEVE_WITHOUT_BULK_DATA_GET	Composite Instance Retrieve Without Bulk Data - GET.
MED_DCM_SOP_MODALITY_WORKLIST_FIND	Modality Worklist Information Model - FIND.
MED_DCM_SOP_GEN_WORKLIST_MANAGEMENT_META	General Purpose Worklist Management Meta SOP Class.
MED_DCM_SOP_GEN_WORKLIST_FIND	General Purpose Worklist Information Model - FIND.
MED_DCM_SOP_GEN_SCHEDULED_PROC_STEP	General Purpose Scheduled Procedure Step.
MED_DCM_SOP_GEN_PERFORMED_PROC_STEP	General Purpose Performed Procedure Step.
MED_DCM_SOP_INSTANCE_AVAILABILITY_NOTIFICATION	Instance Availability Notification SOP Class.
MED_DCM_SOP_RT_BEAMS_DELIVERY_INSTRUCTION_STORAGE	RT Beams Delivery Instruction Storage (Supplement 74 Frozen Draft).
MED_DCM_SOP_RT_CONVENTIONAL_MACHINE_VERIFICATION	RT Conventional Machine Verification (Supplement 74 Frozen Draft).
MED_DCM_SOP_RT_ION_MACHINE_VERIFICATION	RT Ion Machine Verification (Supplement 74 Frozen Draft).
MED_DCM_SOP_UNIFIED_WORKLIST_PROC_STEP_SERVICE_CLASS	Unified Worklist and Procedure Step Service class.
MED_DCM_SOP_UNIFIED_PROC_STEP_PUSH	Unified Procedure Step - Push SOP Class.
MED_DCM_SOP_UNIFIED_PROC_STEP_WATCH	Unified Procedure Step - Watch SOP Class.
MED_DCM_SOP_UNIFIED_PROC_STEP_PULL	Unified Procedure Step - Pull SOP Class.

MED_DCM_SOP_UNIFIED_PROC_STEP_EVENT	Unified Procedure Step - Event SOP Class.
MED_DCM_SOP_UNIFIED_WORKLIST_PROC_STEP_INSTANCE	Unified Worklist and Procedure Step SOP Instance (Well-known).
MED_DCM_SOP_GENERAL_RELEVANT_PATIENT_INFORMATION_QUERY	General Relevant Patient Information Query.
MED_DCM_SOP_BREAST_IMAGING_RELEVANT_PATIENT_INFORMATION_QUERY	Breast Imaging Relevant Patient Information Query.
MED_DCM_SOP_CARDIAC_RELEVANT_PATIENT_INFORMATION_QUERY	Cardiac Relevant Patient Information Query.
MED_DCM_SOP_HANGING_PROTOCOL_STORAGE	Hanging Protocol Storage.
MED_DCM_SOP_HANGING_PROTOCOL_INFORMATION_MODEL_FIND	Hanging Protocol Information Model - FIND.
MED_DCM_SOP_HANGING_PROTOCOL_INFORMATION_MODEL_MOVE	Hanging Protocol Information Model - MOVE.
MED_DCM_SOP_HANGING_PROTOCOL_INFORMATION_MODEL_GET	Hanging Protocol Information Model - GET.
MED_DCM_SOP_COLOR_PALETTE_STORAGE	Color Palette Storage.
MED_DCM_SOP_COLOR_PALETTE_INFORMATION_MODEL_FIND	Color Palette Information Model - FIND.
MED_DCM_SOP_COLOR_PALETTE_INFORMATION_MODEL_MOVE	Color Palette Information Model - MOVE.
MED_DCM_SOP_COLOR_PALETTE_INFORMATION_MODEL_GET	Color Palette Information Model - GET.
MED_DCM_SOP_PRODUCT_CHARACTERISTICS	Product Characteristics Query SOP Class.
MED_DCM_SOP_SUBSTANCE_APPROVAL_QUERY	Substance Approval Query SOP Class.
MED_DCM_SOP_GE_PLAN_STORAGE	GE Plan Storage.
MED_DCM_SOP_GE_MACHINE_STORAGE	GE Machine Storage.
MED_DCM_SOP_MSICOM3_LZW_STORAGE	MsiCOM3 LZW storage.

1.3.2.4.8 enumIGMedTag

Specifies DICOM tag identifiers.

Values:

DCM_TAG_CommandGroupLength	Command Group Length.
DCM_TAG_Group0000Length	Command Group Length. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CommandLengthToEnd	Command Length to End. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group0000LengthToEnd	Command Length to End. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LengthToEnd	Command Length to End. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AffectedSOPClassUID	Affected SOP Class UID.
DCM_TAG_RequestedSOPClassUID	Requested SOP Class UID.
DCM_TAG_RecognitionCode	Recognition Code. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CommandField	Command Field.
DCM_TAG_MessageID	Message ID.
DCM_TAG_MessageIDBeingRespondedTo	Message ID Being Responded To.
DCM_TAG_SenderAeTitle	Initiator. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReceiverAeTitle	Receiver. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_FindLocation	Find Location. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MoveDestination	Move Destination.
DCM_TAG_Priority	Operation Priority.
DCM_TAG_DataSetType	Data Set Type.
DCM_TAG_NumberOfMatches	Number of Matches. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ResponseSequenceNumber	Response Sequence Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Status	Operation Status.

DCM_TAG_OffendingElement	Offending Element.
DCM_TAG_ErrorComment	Error Comment.
DCM_TAG_ErrorID	Error Identifier.
DCM_TAG_AffectedSOPInstanceUID	Affected SOP Instance UID.
DCM_TAG_RequestedSOPInstanceUID	Requested SOP Instance UID.
DCM_TAG_EventTypeID	Event Type ID.
DCM_TAG_AttributeIdentifierList	Attribute Identifier List.
DCM_TAG_ActionTypeID	Action Type ID.
DCM_TAG_NumberOfRemainingSuboperations	Number of Remaining Sub-operations.
DCM_TAG_NumberOfCompletedSuboperations	Number of Completed Sub-operations.
DCM_TAG_NumberOfFailedSuboperations	Number of Failed Sub-operations.
DCM_TAG_NumberOfWarningSuboperations	Number of Warning Sub-operations.
DCM_TAG_MoveOriginatorApplicationEntityTitle	Move Originator Application Entity Title.
DCM_TAG_MoveOriginatorMessageID	Move Originator Message ID.
DCM_TAG_DIALOGReceiver	DIALOG Receiver. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TerminalType	Terminal Type. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MessageSetID	Message Set ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_EndMessageID	End Message ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DisplayFormat	Display Format. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PagePositionID	Page Position ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TextFormatID	Text Format ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NormalReverse	Normal Reverse. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NorRev	Normal Reverse. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AddGrayScale	Add Gray Scale. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.

DCM_TAG_Borders	Borders. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Copies	Copies. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Erase	Erase. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Print	Print. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Overlays	Overlays. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_FileMetaInformationGroupLength	File Meta Information Group Length.
DCM_TAG_Group0002Length	File Meta Information Group Length. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FileMetaInformationVersion	File Meta Information Version.
DCM_TAG_MediaStorageSOPClassUID	Media Storage SOP Class UID.
DCM_TAG_MediaStoredSOPClassUID	Media Storage SOP Class UID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MediaStorageSOPInstanceUID	Media Storage SOP Instance UID.
DCM_TAG_MediaStoredSOPInstanceUID	Media Storage SOP Instance UID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TransferSyntaxUID	Transfer Syntax UID. This tells whether the DICOM file uses big or little endian format, which compression is used, if any, and whether the file uses Implicit or Explicit Value Representation syntax.
DCM_TAG_ImplementationClassUID	Implementation Class UID.
DCM_TAG_ImplementationVersionName	Implementation Version Name.
DCM_TAG_SourceApplicationEntityTitle	Source Application Entity Title.
DCM_TAG_PrivateInformationCreatorUID	Private Information Creator UID.
DCM_TAG_PrivateInformation	Private Information.
DCM_TAG_Group0004Length	Group 0004 Length. This tag is marked as retired in DICOM

	specification. See DICOM specification for alternatives.
DCM_TAG_FilesetID	File-set ID.
DCM_TAG_FilesetDescriptorFileID	File-set Descriptor File ID.
DCM_TAG_SpecificCharacterSetOfFilesetDescriptorFile	Specific Character Set of File-set Descriptor File.
DCM_TAG_CharSet	Specific Character Set of File-set Descriptor File. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OffsetOfTheFirstDirectoryRecordOfTheRootDirectoryEntity	Offset of the First Directory Record of the Root Directory Entity.
DCM_TAG_RootDirectoryEntitysFirstDirectoryRecordOffset	Offset of the First Directory Record of the Root Directory Entity. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OffsetOfTheLastDirectoryRecordOfTheRootDirectoryEntity	Offset of the Last Directory Record of the Root Directory Entity.
DCM_TAG_RootDirectoryEntitysLastDirectoryRecordOffset	Offset of the Last Directory Record of the Root Directory Entity. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FilesetConsistencyFlag	File-set Consistency Flag.
DCM_TAG_DirectoryRecordSequence	Directory Record Sequence.
DCM_TAG_OffsetOfTheNextDirectoryRecord	Offset of the Next Directory Record.
DCM_TAG_NextDirectoryRecordOffset	Offset of the Next Directory Record. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RecordInUseFlag	Record In-use Flag.
DCM_TAG_OffsetOfReferencedLowerLevelDirectoryEntity	Offset of Referenced Lower-Level Directory Entity.
DCM_TAG_ReferencedLowerlevelDirectoryEntityOffset	Offset of Referenced Lower-Level Directory Entity. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DirectoryRecordType	Directory Record Type.
DCM_TAG_PrivateRecordUID	Private Record UID.
DCM_TAG_ReferencedFileID	Referenced File ID.
DCM_TAG_MRDRDirectoryRecordOffset	MRDR Directory Record Offset.

	This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedSOPClassUIDInFile	Referenced SOP Class UID in File.
DCM_TAG_ReferencedSOPInstanceUIDInFile	Referenced SOP Instance UID in File.
DCM_TAG_ReferencedTransferSyntaxUIDInFile	Referenced Transfer Syntax UID in File.
DCM_TAG_ReferencedFileXferSynUID	Referenced Transfer Syntax UID in File.
DCM_TAG_ReferencedRelatedGeneralSOPClassUIDinFile	Referenced Related General SOP Class UID in File.
DCM_TAG_NumberOfReferences	Number of References. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group0008Length	Group 0008 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group0008LengthToEnd	Group 0008 Length to End. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SpecificCharacterSet	Specific Character Set.
DCM_TAG_LanguageCodeSequence	Language Code Sequence.
DCM_TAG_ImageType	Image Type.
DCM_TAG_RecognitionCodeRetired	Recognition Code (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InstanceCreationDate	Instance Creation Date.
DCM_TAG_InstanceCreationTime	Instance Creation Time.
DCM_TAG_InstanceCreatorUID	Instance Creator UID.
DCM_TAG_SOPClassUID	SOP Class UID.
DCM_TAG_SOPInstanceUID	SOP Instance UID.
DCM_TAG_RelatedGeneralSOPClassUID	Related General SOP Class UID.
DCM_TAG_RelatedGeneral	Related General SOP Class UID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OriginalSpecializedSOPClassUID	Original Specialized SOP Class UID.
DCM_TAG_OriginalSpecialized	Original Specialized SOP Class UID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_StudyDate	Study Date.
DCM_TAG_SeriesDate	Series Date.
DCM_TAG_AcquisitionDate	Acquisition Date.
DCM_TAG_ContentDate	Content Date.

DCM_TAG_OverlayDate	Overlay Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveDate	Curve Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AcquisitionDatetime	Acquisition DateTime. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_StudyTime	Study Time.
DCM_TAG_SeriesTime	Series Time.
DCM_TAG_AcquisitionTime	Acquisition Time.
DCM_TAG_ContentTime	Content Time.
DCM_TAG_OverlayTime	Overlay Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveTime	Curve Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DataSetSubtype	Data Set Subtype. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NuclearMedicineSeriesType	Nuclear Medicine Series Type. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AccessionNumber	Accession Number.
DCM_TAG_IssuerOfAccessionNumberSequence	Issuer of Accession Number Sequence.
DCM_TAG_QueryRetrieveLevel	Query/Retrieve Level.
DCM_TAG_RetrieveAETitle	Retrieve AE Title.
DCM_TAG_InstanceAvailability	Instance Availability.
DCM_TAG_FailedSOPInstanceUIDList	Failed SOP Instance UID List.
DCM_TAG_Modality	Modality value.
DCM_TAG_ModalitiesInStudy	Modalities in Study.
DCM_TAG_SOPClassesInStudy	SOP Classes In Study.
DCM_TAG_ConversionType	Conversion Type.
DCM_TAG_PresentationIntentType	Presentation Intent Type.
DCM_TAG_Manufacturer	The Manufacturer.
DCM_TAG_InstitutionName	Institution Name.
DCM_TAG_InstitutionAddress	Institution Address.
DCM_TAG_InstitutionCodeSequence	Institution Code Sequence.
DCM_TAG_ReferringPhysicianName	Referring Physician's Name.
DCM_TAG_ReferringPhysiciansName	Referring Physician's Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous

	line.
DCM_TAG_ReferringPhysicianAddress	Referring Physician's Address.
DCM_TAG_ReferringPhysiciansAddress	Referring Physician's Address. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferringPhysicianTelephoneNumbers	Referring Physician's Telephone Numbers.
DCM_TAG_ReferringPhysiciansTelephoneNumbers	Referring Physician's Telephone Numbers. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferringPhysicianIdentificationSequence	Referring Physician Identification Sequence.
DCM_TAG_ReferringPhysicianIDSequence	Referring Physician Identification Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CodeValue	Code Value.
DCM_TAG_CodingSchemeDesignator	Coding Scheme Designator.
DCM_TAG_CodingSchemeVersion	Coding Scheme Version.
DCM_TAG_CodeMeaning	Code Meaning.
DCM_TAG_MappingResource	Mapping Resource.
DCM_TAG_ContextGroupVersion	Context Group Version.
DCM_TAG_ContextGroupLocalVersion	Context Group Local Version.
DCM_TAG_ContextGroupExtensionFlag	Context Group Extension Flag.
DCM_TAG_CodingSchemeUID	Coding Scheme UID.
DCM_TAG_ContextGroupExtensionCreatorUID	Context Group Extension Creator UID.
DCM_TAG_ContextIdentifier	Context Identifier.
DCM_TAG_CodingSchemeIdentificationSequence	Coding Scheme Identification Sequence.
DCM_TAG_CodingSchemeIDSequence	Coding Scheme Identification Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CodingSchemeRegistry	Coding Scheme Registry.
DCM_TAG_CodingSchemeExternalID	Coding Scheme External ID.
DCM_TAG_CodingSchemeName	Coding Scheme Name.
DCM_TAG_CodingSchemeResponsibleOrganization	Coding Scheme Responsible Organization.
DCM_TAG_ContextUID	Context UID.
DCM_TAG_TimezoneOffsetFromUTC	Timezone Offset From UTC.
DCM_TAG_NetworkID	Network ID. This tag is marked as retired in DICOM specification.

	See DICOM specification for alternatives.
DCM_TAG_StationName	Station Name.
DCM_TAG_StudyDescription	Study Description.
DCM_TAG_ProcedureCodeSequence	Procedure Code Sequence.
DCM_TAG_SeriesDescription	Series Description.
DCM_TAG_SeriesDescriptionCodeSequence	Series Description Code Sequence.
DCM_TAG_InstitutionalDepartmentName	Institutional Department Name.
DCM_TAG_PhysiciansOfRecord	Physician(s) of Record.
DCM_TAG_PhysiciansOfRecordIdentificationSequence	Physician(s) of Record Identification Sequence.
DCM_TAG_PhysicianOfRecordIDSequence	Physician(s) of Record Identification Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PerformingPhysicianName	Performing Physician's Name.
DCM_TAG_PerformingPhysiciansName	Performing Physician's Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PerformingPhysicianIdentificationSequence	Performing Physician Identification Sequence.
DCM_TAG_PerformingPhysicianIDSequence	Performing Physician Identification Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_NameOfPhysiciansReadingStudy	Name of Physician(s) Reading Study.
DCM_TAG_PhysiciansReadingStudyIdentificationSequence	Physician(s) Reading Study Identification Sequence.
DCM_TAG_PhysicianReadingStudyIDSequence	Physician(s) Reading Study Identification Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OperatorsName	Operators' Name.
DCM_TAG_OperatorIdentificationSequence	Operator Identification Sequence.
DCM_TAG_OperatorIDSequence	Operator Identification Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AdmittingDiagnosesDescription	Admitting Diagnoses Description.
DCM_TAG_AdmittingDiagnosesCodeSequence	Admitting Diagnoses Code Sequence.

DCM_TAG_AdmittingDiagnosisCodeSequence	Admitting Diagnoses Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ManufacturerModelName	Manufacturer's Model Name.
DCM_TAG_ManufacturersModelName	Manufacturer's Model Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedResultsSequence	Referenced Results Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedStudySequence	Referenced Study Sequence.
DCM_TAG_ReferencedPerformedProcedureStepSequence	Referenced Performed Procedure Step Sequence.
DCM_TAG_ReferencedPerformedProcStepSequence	Referenced Performed Procedure Step Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedSeriesSequence	Referenced Series Sequence.
DCM_TAG_ReferencedPatientSequence	Referenced Patient Sequence.
DCM_TAG_ReferencedVisitSequence	Referenced Visit Sequence.
DCM_TAG_ReferencedOverlaySequence	Referenced Overlay Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedStereometricInstanceSequence	Referenced Stereometric Instance Sequence.
DCM_TAG_ReferencedWaveformSequence	Referenced Waveform Sequence.
DCM_TAG_ReferencedImageSequence	Referenced Image Sequence.
DCM_TAG_ReferencedCurveSequence	Referenced Curve Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedPreviousWaveform	Referenced Previous Waveform.
DCM_TAG_ReferencedInstanceSequence	Referenced Instance Sequence.
DCM_TAG_ReferencedSimultaneousWaveforms	Referenced Instance Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedRealWorldValueMappingInstanceSequence	Referenced Real World Value Mapping Instance Sequence.
DCM_TAG_ReferencedRealWorldValueMappingInstance	Referenced Real World Value Mapping Instance Sequence. This tag name has been deprecated and will be removed from the public API in a future release.

	Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedSubsequentWaveform	Referenced Subsequent Waveform. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedSOPClassUID	Referenced SOP Class UID.
DCM_TAG_ReferencedSOPInstanceUID	Referenced SOP Instance UID.
DCM_TAG_SOPClassesSupported	SOP Classes Supported.
DCM_TAG_SOPClassSupported	SOP Classes Supported. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedFrameNumber	Referenced Frame Number.
DCM_TAG_SimpleFrameList	Simple Frame List.
DCM_TAG_CalculatedFrameList	Calculated Frame List.
DCM_TAG_TimeRange	Time Range.
DCM_TAG_FrameExtractionSequence	Frame Extraction Sequence.
DCM_TAG_MultiFrameSourceSOPInstanceUID	Multi-Frame Source SOP Instance UID.
DCM_TAG_TransactionUID	Transaction UID.
DCM_TAG_FailureReason	Failure Reason.
DCM_TAG_FailedSOPSequence	Failed SOP Sequence.
DCM_TAG_ReferencedSOPSequence	Referenced SOP Sequence.
DCM_TAG_StudiesContainingOtherReferencedInstancesSequence	Studies Containing Other Referenced Instances Sequence.
DCM_TAG_RelatedSeriesSequence	Related Series Sequence.
DCM_TAG_LossyImageCompressionRetired	Lossy Image Compression (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DerivationDescription	Derivation Description.
DCM_TAG_SourceImageSequence	Source Image Sequence.
DCM_TAG_StageName	Stage Name.
DCM_TAG_StageNumber	Stage Number.
DCM_TAG_NumberOfStages	Number of Stages.
DCM_TAG_ViewName	View Name.
DCM_TAG_ViewNumber	View Number.
DCM_TAG_NumberOfEventTimers	Number of Event Timers.
DCM_TAG_NumberOfViewsInStage	Number of Views in Stage.
DCM_TAG_EventElapsedTimes	Event Elapsed Time(s).
DCM_TAG_EventTimerNames	Event Timer Name(s).
DCM_TAG_EventTimerSequence	Event Timer Sequence.
DCM_TAG_EventTimeOffset	Event Time Offset.
DCM_TAG_EventCodeSequence	Event Code Sequence.
DCM_TAG_StartTrim	Start Trim.
DCM_TAG_StopTrim	Stop Trim.
DCM_TAG_RecommendedDisplayFrameRate	Recommended Display Frame

	Rate.
DCM_TAG_TransducerPosition	Transducer Position.
DCM_TAG_TransducerOrientation	Transducer Orientation.
DCM_TAG_AnatomicStructure	Anatomic Structure.
DCM_TAG_AnatomicRegionSequence	Anatomic Region Sequence.
DCM_TAG_AnatomicRegionModifierSequence	Anatomic Region Modifier Sequence.
DCM_TAG_PrimaryAnatomicStructureSequence	Primary Anatomic Structure Sequence.
DCM_TAG_AnatomicStructureSpaceOrRegionSequence	Anatomic Structure, Space or Region Sequence.
DCM_TAG_AnatomicStructureSpaceRegionSequence	Anatomic Structure, Space or Region Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PrimaryAnatomicStructureModifierSequence	Primary Anatomic Structure Modifier Sequence.
DCM_TAG_TransducerPositionSequence	Transducer Position Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TransducerPositionModifierSequence	Transducer Position Modifier Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TransducerOrientationSequence	Transducer Orientation Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TransducerOrientationModifierSequence	Transducer Orientation Modifier Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnatomicStructureSpaceOrRegionCodeSequence	Anatomic Structure Space Or Region Code Sequence (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnatomicPortalOfEntranceCodeSequence	Anatomic Portal Of Entrance Code Sequence (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnatomicApproachDirectionCodeSequence	Anatomic Approach Direction Code Sequence (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnatomicPerspectiveDescription	Anatomic Perspective Description (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnatomicPerspectiveCodeSequence	Anatomic Perspective Code Sequence (Trial). This tag is

	marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnatomicLocationOfExaminingInstrumentDescription	Anatomic Location Of Examining Instrument Description (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnatomicLocationOfExaminingInstrumentCodeSequence	Anatomic Location Of Examining Instrument Code Sequence (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnatomicStructureSpaceOrRegionModifierCodeSequence	Anatomic Structure Space Or Region Modifier Code Sequence (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OnAxisBackgroundAnatomicStructureCodeSequence	OnAxis Background Anatomic Structure Code Sequence (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AlternateRepresentationSequence	Alternate Representation Sequence.
DCM_TAG_IrradiationEventUID	Irradiation Event UID.
DCM_TAG_IdentifyingComments	Identifying Comments. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_FrameType	Frame Type.
DCM_TAG_ReferencedImageEvidenceSequence	Referenced Image Evidence Sequence.
DCM_TAG_RefImgEvidenceSequence	Referenced Image Evidence Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedRawDataSequence	Referenced Raw Data Sequence.
DCM_TAG_RefRawDataSequence	Referenced Raw Data Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CreatorVersionUID	Creator Version UID.
DCM_TAG_DerivationImageSequence	Derivation Image Sequence.
DCM_TAG_DerivationImgSequence	Derivation Image Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SourceImageEvidenceSequence	Source Image Evidence Sequence.

DCM_TAG_SrcImgEvidenceSequence	Source Image Evidence Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PixelPresentation	Pixel Presentation.
DCM_TAG_VolumetricProperties	Volumetric Properties.
DCM_TAG_VolumeBasedCalculationTechnique	Volume Based Calculation Technique.
DCM_TAG_VolumeBasedCalcTechnique	Volume Based Calculation Technique. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ComplexImageComponent	Complex Image Component.
DCM_TAG_ComplexImgComponent	Complex Image Component. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AcquisitionContrast	Acquisition Contrast.
DCM_TAG_DerivationCodeSequence	Derivation Code Sequence.
DCM_TAG_ReferencedPresentationStateSequence	Referenced Presentation State Sequence.
DCM_TAG_ReferencedGrayscalePresentationStateSequence	Referenced Grayscale Presentation State Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RefGrayscalePresStateSequence	Referenced Grayscale Presentation State Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedOtherPlaneSequence	Referenced Other Plane Sequence.
DCM_TAG_FrameDisplaySequence	Frame Display Sequence.
DCM_TAG_RecommendedDisplayFrameRateinFloat	Recommended Display Frame Rate in Float.
DCM_TAG_SkipFrameRangeFlag	Skip Frame Range Flag.
DCM_TAG_Group0010Length	Group 0010 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PatientName	Patient's Name.
DCM_TAG_PatientsName	Patient's Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the

	tag with the same value defined in the previous line.
DCM_TAG_PatientID	Patient ID.
DCM_TAG_IssuerOfPatientID	Issuer of Patient ID.
DCM_TAG_TypeOfPatientID	Type of Patient ID.
DCM_TAG_IssuerOfPatientIDQualifiersSequence	Issuer of Patient ID Qualifiers Sequence.
DCM_TAG_PatientBirthDate	Patient's Birth Date.
DCM_TAG_PatientsBirthDate	Patient's Birth Date. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientBirthTime	Patient's Birth Time.
DCM_TAG_PatientsBirthTime	Patient's Birth Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientSex	Patient's Sex.
DCM_TAG_PatientsSex	Patient's Sex. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientInsurancePlanCodeSequence	Patient's Insurance Plan Code Sequence.
DCM_TAG_PatientsInsurancePlanCodeSequence	Patient's Insurance Plan Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientPrimaryLanguageCodeSequence	Patient's Primary Language Code Sequence.
DCM_TAG_PatientsPrimaryLanguageCodeSequence	Patient's Primary Language Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientPrimaryLangCodeSequence	Patient's Primary Language Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientPrimaryLanguageModifierCodeSequence	Patient's Primary Language Modifier Code Sequence.
DCM_TAG_PatientsPrimaryLanguageCodeModifierSequence	Patient's Primary Language Code Modifier Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value

	defined in the previous line.
DCM_TAG_PatientPrimaryLangCodeModSequence	Patient's Primary Language Code Modifier Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OtherPatientIDs	Other Patient IDs.
DCM_TAG_OtherPatientNames	Other Patient Names.
DCM_TAG_OtherPatientIDsSequence	Other Patient IDs Sequence.
DCM_TAG_PatientBirthName	Patient's Birth Name.
DCM_TAG_PatientsBirthName	Patient's Birth Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientAge	Patient's Age.
DCM_TAG_PatientsAge	Patient's Age. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientSize	Patient's Size.
DCM_TAG_PatientsSize	Patient's Size. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientWeight	Patient's Weight.
DCM_TAG_PatientsWeight	Patient's Weight. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientAddress	Patient's Address.
DCM_TAG_PatientsAddress	Patient's Address. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_InsurancePlanIdentification	Insurance Plan Identification. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PatientMotherBirthName	Patient's Mother's Birth Name.
DCM_TAG_PatientsMothersBirthName	Patient's Mother's Birth Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MilitaryRank	Military Rank.
DCM_TAG_BranchOfService	Branch of Service.

DCM_TAG_MedicalRecordLocator	Medical Record Locator.
DCM_TAG_MedicalAlerts	Medical Alerts.
DCM_TAG_Allergies	Allergies value.
DCM_TAG_ContrastAllergies	Allergies value. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CountryOfResidence	Country of Residence.
DCM_TAG_RegionOfResidence	Region of Residence.
DCM_TAG_PatientTelephoneNumbers	Patient's Telephone Numbers.
DCM_TAG_PatientsTelephoneNumbers	Patient's Telephone Numbers. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_EthnicGroup	Ethnic Group.
DCM_TAG_Occupation	Patient's Occupation.
DCM_TAG_SmokingStatus	Smoking Status.
DCM_TAG_AdditionalPatientHistory	Additional Patient History.
DCM_TAG_PregnancyStatus	Pregnancy Status.
DCM_TAG_LastMenstrualDate	Last Menstrual Date.
DCM_TAG_PatientReligiousPreference	Patient's Religious Preference.
DCM_TAG_PatientsReligiousPreference	Patient's Religious Preference. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientSpeciesDescription	Patient Species Description.
DCM_TAG_PatientSpeciesCodeSequence	Patient Species Code Sequence.
DCM_TAG_PatientSexNeutered	Patient's Sex Neutered.
DCM_TAG_PatientsSexNeutered	Patient's Sex Neutered. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AnatomicalOrientationType	Anatomical Orientation Type.
DCM_TAG_PatientBreedDescription	Patient Breed Description.
DCM_TAG_PatientBreedCodeSequence	Patient Breed Code Sequence.
DCM_TAG_BreedRegistrationSequence	Breed Registration Sequence.
DCM_TAG_BreedRegistrationNumber	Breed Registration Number.
DCM_TAG_BreedRegistryCodeSequence	Breed Registry Code Sequence.
DCM_TAG_ResponsiblePerson	Responsible Person.
DCM_TAG_ResponsiblePersonRole	Responsible Person Role.
DCM_TAG_ResponsibleOrganization	Responsible Organization.
DCM_TAG_PatientResponsibleOrganization	Responsible Organization. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same

	value defined in the previous line.
DCM_TAG_PatientComments	Patient Comments.
DCM_TAG_ExaminedBodyThickness	Examined Body Thickness.
DCM_TAG_Group0012Length	Group 0012 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ClinicalTrialSponsorName	Clinical Trial Sponsor Name.
DCM_TAG_ClinicalTrialProtocolID	Clinical Trial Protocol ID.
DCM_TAG_ClinicalTrialProtocolName	Clinical Trial Protocol Name.
DCM_TAG_ClinicalTrialSiteID	Clinical Trial Site ID.
DCM_TAG_ClinicalTrialSiteName	Clinical Trial Site Name.
DCM_TAG_ClinicalTrialSubjectID	Clinical Trial Subject ID.
DCM_TAG_ClinicalTrialSubjectReadingID	Clinical Trial Subject Reading ID.
DCM_TAG_ClinicalTrialTimePointID	Clinical Trial Time Point ID.
DCM_TAG_ClinicalTrialTimePointDescription	Clinical Trial Time Point Description.
DCM_TAG_ClinicalTrialTimePointDesc	Clinical Trial Time Point Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ClinicalTrialCoordinatingCenterName	Clinical Trial Coordinating Center Name.
DCM_TAG_ClinicalTrialCenterName	Clinical Trial Coordinating Center Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientIdentityRemoved	Patient Identity Removed.
DCM_TAG_DeidentificationMethod	De-identification Method.
DCM_TAG_DeidentificationMethodCodeSequence	De-identification Method Code Sequence.
DCM_TAG_ClinicalTrialSeriesID	Clinical Trial Series ID.
DCM_TAG_ClinicalTrialSeriesDescription	Clinical Trial Series Description.
DCM_TAG_ClinicalTrialProtocolEthicsCommitteeName	Clinical Trial Protocol Ethics Committee Name.
DCM_TAG_ClinicalTrialProtocolEthicsCommitteeApprovalNumber	Clinical Trial Protocol Ethics Committee Approval Number.
DCM_TAG_ConsentForClinicalTrialUseSequence	Consent for Clinical Trial Use Sequence.
DCM_TAG_DistributionType	Distribution Type.
DCM_TAG_ConsentForDistributionFlag	Consent for Distribution Flag.
DCM_TAG_Group0018Length	Group 0018 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ContrastBolusAgent	Contrast/Bolus Agent.
DCM_TAG_ContrastBolusAgentSequence	Contrast/Bolus Agent Sequence.
DCM_TAG_ContrastBolusAdministrationRouteSequence	Contrast/Bolus Administration Route Sequence.

DCM_TAG_ContrastBolusAdminRouteSequence	Contrast/Bolus Administration Route Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BodyPartExamined	Body Part Examined.
DCM_TAG_ScanningSequence	Scanning Sequence.
DCM_TAG_SequenceVariant	Sequence Variant.
DCM_TAG_ScanOptions	Scan Options.
DCM_TAG_MrAcquisitionType	MR Acquisition Type.
DCM_TAG_SequenceName	Sequence Name.
DCM_TAG_AngioFlag	Angio Flag.
DCM_TAG_InterventionDrugInformationSequence	Intervention Drug Information Sequence.
DCM_TAG_InterventionDrugInfoSequence	Intervention Drug Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_InterventionDrugStopTime	Intervention Drug Stop Time.
DCM_TAG_InterventionDrugDose	Intervention Drug Dose.
DCM_TAG_InterventionDrugCodeSequence	Intervention Drug Code Sequence.
DCM_TAG_InterventionDrugSequence	Intervention Drug Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AdditionalDrugSequence	Additional Drug Sequence.
DCM_TAG_Radionuclide	Radionuclide. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Radiopharmaceutical	Radiopharmaceutical value.
DCM_TAG_EnergyWindowCenterline	Energy Window Centerline. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_EnergyWindowTotalWidth	Energy Window Total Width. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterventionDrugName	Intervention Drug Name.
DCM_TAG_InterventionDrugStartTime	Intervention Drug Start Time.
DCM_TAG_InterventionSequence	Intervention Sequence.
DCM_TAG_TherapyType	Therapy Type. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterventionStatus	Intervention Status.
DCM_TAG_TherapyDescription	Therapy Description. This tag is marked as retired in DICOM

	specification. See DICOM specification for alternatives.
DCM_TAG_InterventionDescription	Intervention Description.
DCM_TAG_CineRate	Cine Rate.
DCM_TAG_InitialCineRunState	Initial Cine Run State.
DCM_TAG_SliceThickness	Slice Thickness.
DCM_TAG_Kvp	KVP (kilovolts peak).
DCM_TAG_CountsAccumulated	Counts Accumulated.
DCM_TAG_AcquisitionTerminationCondition	Acquisition Termination Condition.
DCM_TAG_EffectiveDuration	Effective Duration.
DCM_TAG_EffectiveSeriesDuration	Effective Duration. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AcquisitionStartCondition	Acquisition Start Condition.
DCM_TAG_AcqStartCondition	Acquisition Start Condition. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AcquisitionStartConditionData	Acquisition Start Condition Data.
DCM_TAG_AcqStartConditionData	Acquisition Start Condition Data. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AcquisitionTerminationConditionData	Acquisition Termination Condition Data.
DCM_TAG_AcqStopConditionData	Acquisition Termination Condition Data. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RepetitionTime	Repetition Time.
DCM_TAG_EchoTime	Echo Time.
DCM_TAG_InversionTime	Inversion Time.
DCM_TAG_NumberOfAverages	Number of Averages.
DCM_TAG_ImagingFrequency	Imaging Frequency.
DCM_TAG_ImagedNucleus	Imaged Nucleus.
DCM_TAG_EchoNumbers	Echo Number(s).
DCM_TAG_MagneticFieldStrength	Magnetic Field Strength.
DCM_TAG_SpacingBetweenSlices	Spacing Between Slices.
DCM_TAG_NumberOfPhaseEncodingSteps	Number of Phase Encoding Steps.
DCM_TAG_DataCollectionDiameter	Data Collection Diameter.
DCM_TAG_EchoTrainLength	Echo Train Length.
DCM_TAG_PercentSampling	Percent Sampling.

DCM_TAG_PercentPhaseFieldOfView	Percent Phase Field of View.
DCM_TAG_PixelBandwidth	Pixel Bandwidth.
DCM_TAG_DeviceSerialNumber	Device Serial Number.
DCM_TAG_DeviceUID	Device UID.
DCM_TAG_DeviceID	Device ID.
DCM_TAG_PlateID	Plate Identifier.
DCM_TAG_GeneratorID	Generator ID.
DCM_TAG_GridID	Grid Identifier.
DCM_TAG_CassetteID	Cassette ID.
DCM_TAG_GantryID	Gantry ID.
DCM_TAG_SecondaryCaptureDeviceID	Secondary Capture Device ID.
DCM_TAG_HardcopyCreationDeviceID	Hardcopy Creation Device ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DateOfSecondaryCapture	Date of Secondary Capture.
DCM_TAG_TimeOfSecondaryCapture	Time of Secondary Capture.
DCM_TAG_SecondaryCaptureDeviceManufacturer	Secondary Capture Device Manufacturer.
DCM_TAG_SecondaryCaptureDeviceManufacturers	Secondary Capture Device Manufacturers. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_HardcopyDeviceManufacturer	Hardcopy Device Manufacturer. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SecondaryCaptureDeviceManufacturerModelName	Secondary Capture Device Manufacturer's Model Name.
DCM_TAG_SecondaryCaptureDeviceManufacturersModelName	Secondary Capture Device Manufacturer's Model Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SecondaryCaptureDeviceSoftwareVersions	Secondary Capture Device Software Version(s).
DCM_TAG_HardcopyDeviceSoftwareVersion	Hardcopy Device Software Version. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_HardcopyDeviceManufacturersModelName	Hardcopy Device Manufacturer's Model Name. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SoftwareVersions	Software Version(s).
DCM_TAG_VideoImageFormatAcquired	Video Image Format Acquired.
DCM_TAG_DigitalImageFormatAcquired	Digital Image Format Acquired.
DCM_TAG_ProtocolName	Protocol Name.
DCM_TAG_ContrastBolusRoute	Contrast/Bolus Route.

DCM_TAG_ContrastBolusVolume	Contrast/Bolus Volume.
DCM_TAG_ContrastBolusStartTime	Contrast/Bolus Start Time.
DCM_TAG_ContrastBolusStopTime	Contrast/Bolus Stop Time.
DCM_TAG_ContrastBolusTotalDose	Contrast/Bolus Total Dose.
DCM_TAG_SyringeCounts	Syringe Counts.
DCM_TAG_ContrastFlowRate	Contrast Flow Rate.
DCM_TAG_ContrastFlowDuration	Contrast Flow Duration.
DCM_TAG_ContrastBolusIngredient	Contrast/Bolus Ingredient.
DCM_TAG_ContrastBolusIngredientConcentration	Contrast/Bolus Ingredient Concentration.
DCM_TAG_SpatialResolution	Spatial Resolution.
DCM_TAG_TriggerTime	Trigger Time.
DCM_TAG_TriggerSourceorType	Trigger Source or Type.
DCM_TAG_NominalInterval	Nominal Interval.
DCM_TAG_FrameTime	Frame Time.
DCM_TAG_CardiacFramingType	Cardiac Framing Type.
DCM_TAG_FramingType	Cardiac Framing Type. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FrameTimeVector	Frame Time Vector.
DCM_TAG_FrameDelay	Frame Delay.
DCM_TAG_ImageTriggerDelay	Image Trigger Delay.
DCM_TAG_MultiplexGroupTimeOffset	Multiplex Group Time Offset.
DCM_TAG_TriggerTimeOffset	Trigger Time Offset.
DCM_TAG_SynchronizationTrigger	Synchronization Trigger.
DCM_TAG_SynchronizationChannel	Synchronization Channel.
DCM_TAG_TriggerSamplePosition	Trigger Sample Position.
DCM_TAG_RadiopharmaceuticalRoute	Radiopharmaceutical Route.
DCM_TAG_RadiopharmaRoute	Radiopharmaceutical Route. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RadiopharmaceuticalVolume	Radiopharmaceutical Volume.
DCM_TAG_RadiopharmaVolume	Radiopharmaceutical Volume. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RadiopharmaceuticalStartTime	Radiopharmaceutical Start Time.
DCM_TAG_RadiopharmaStartTime	Radiopharmaceutical Start Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RadiopharmaceuticalStopTime	Radiopharmaceutical Stop Time.

DCM_TAG_RadiopharmaStopTime	Radiopharmaceutical Stop Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RadionuclideTotalDose	Radionuclide Total Dose.
DCM_TAG_RadionuclideHalfLife	Radionuclide Half Life.
DCM_TAG_RadionuclidePositronFraction	Radionuclide Positron Fraction.
DCM_TAG_RadiopharmaceuticalSpecificActivity	Radiopharmaceutical Specific Activity.
DCM_TAG_RadiopharmaSpecificActivity	Radiopharmaceutical Specific Activity. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RadiopharmaceuticalStartDatetime	Radiopharmaceutical Start DateTime. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RadiopharmaceuticalStopDatetime	Radiopharmaceutical Stop DateTime. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BeatRejectionFlag	Beat Rejection Flag.
DCM_TAG_LowRrValue	Low R-R Value.
DCM_TAG_HighRrValue	High R-R Value.
DCM_TAG_IntervalsAcquired	Intervals Acquired.
DCM_TAG_IntervalsRejected	Intervals Rejected.
DCM_TAG_PvcRejection	PVC Rejection.
DCM_TAG_SkipBeats	Skip Beats.
DCM_TAG_HeartRate	Heart Rate.
DCM_TAG_CardiacNumberOfImages	Cardiac Number of Images.
DCM_TAG_TriggerWindow	Trigger Window.
DCM_TAG_ReconstructionDiameter	Reconstruction Diameter.
DCM_TAG_DistanceSourceToDetector	Distance Source to Detector.
DCM_TAG_DistanceSourceToPatient	Distance Source to Patient.
DCM_TAG_EstimatedRadiographicMagnificationFactor	Estimated Radiographic Magnification Factor.
DCM_TAG_EstRadiographicMagFactor	Estimated Radiographic Magnification Factor. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_GantryDetectorTilt	Gantry/Detector Tilt.
DCM_TAG_GantryDetectorSlew	Gantry/Detector Slew.
DCM_TAG_TableHeight	Table Height.

DCM_TAG_TableTraverse	Table Traverse.
DCM_TAG_TableMotion	Table Motion.
DCM_TAG_TableVerticalIncrement	Table Vertical Increment.
DCM_TAG_TableLateralIncrement	Table Lateral Increment.
DCM_TAG_TableLongitudinalIncrement	Table Longitudinal Increment.
DCM_TAG_TableAngle	Table Angle.
DCM_TAG_TableType	Table Type.
DCM_TAG_RotationDirection	Rotation Direction.
DCM_TAG_AngularPosition	Angular Position. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RadialPosition	Radial Position.
DCM_TAG_ScanArc	Scan Arc value.
DCM_TAG_AngularStep	Angular Step.
DCM_TAG_CenterOfRotationOffset	Center of Rotation Offset.
DCM_TAG_RotationOffset	Rotation Offset.
DCM_TAG_FieldOfViewShape	Field of View Shape.
DCM_TAG_FieldOfViewDimensions	Field of View Dimension(s).
DCM_TAG_ExposureTime	Exposure Time.
DCM_TAG_XrayTubeCurrent	X-Ray Tube Current. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_Exposure	Exposure value.
DCM_TAG_ExposureInuAs	Exposure in micro As.
DCM_TAG_ExposureInMicroAs	Exposure in micro As. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ExposureInMicroA	Exposure in micro As. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AveragePulseWidth	Average Pulse Width.
DCM_TAG_RadiationSetting	Radiation Setting.
DCM_TAG_RectificationType	Rectification Type.
DCM_TAG_RadiationMode	Radiation Mode.
DCM_TAG_ImageAndFluoroscopyAreaDoseProduct	Image and Fluoroscopy Area Dose Product.
DCM_TAG_ImageAreaDoseProduct	Image and Fluoroscopy Area Dose Product. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FilterType	Filter Type.
DCM_TAG_TypeOfFilters	Type of Filters.

DCM_TAG_IntensifierSize	Intensifier Size.
DCM_TAG_ImagerPixelSpacing	Imager Pixel Spacing.
DCM_TAG_Grid	Grid value.
DCM_TAG_GeneratorPower	Generator Power.
DCM_TAG_CollimatorGridName	Collimator/grid Name.
DCM_TAG_CollimatorType	Collimator Type.
DCM_TAG_FocalDistance	Focal Distance.
DCM_TAG_XFocusCenter	X Focus Center.
DCM_TAG_YFocusCenter	Y Focus Center.
DCM_TAG_FocalSpots	Focal Spot(s).
DCM_TAG_AnodeTargetMaterial	Anode Target Material.
DCM_TAG_BodyPartThickness	Body Part Thickness.
DCM_TAG_CompressionForce	Compression Force.
DCM_TAG_DateOfLastCalibration	Date of Last Calibration.
DCM_TAG_TimeOfLastCalibration	Time of Last Calibration.
DCM_TAG_ConvolutionKernel	Convolution Kernel.
DCM_TAG_UpperLowerPixelValues	Upper/Lower Pixel Values. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ActualFrameDuration	Actual Frame Duration.
DCM_TAG_CountRate	Count Rate.
DCM_TAG_PreferredPlaybackSequencing	Preferred Playback Sequencing.
DCM_TAG_ReceiveCoilName	Receive Coil Name.
DCM_TAG_TransmitCoilName	Transmit Coil Name.
DCM_TAG_PlateType	Plate Type.
DCM_TAG_PhosphorType	Phosphor Type.
DCM_TAG_ScanVelocity	Scan Velocity.
DCM_TAG_WholeBodyTechnique	Whole Body Technique.
DCM_TAG_ScanLength	Scan Length.
DCM_TAG_AcquisitionMatrix	Acquisition Matrix.
DCM_TAG_InPlanePhaseEncodingDirection	In-plane Phase Encoding Direction.
DCM_TAG_FlipAngle	Flip Angle.
DCM_TAG_VariableFlipAngleFlag	Variable Flip Angle Flag.
DCM_TAG_Sar	SAR (specific absorption rate).
DCM_TAG_DbDt	The dB/dt.
DCM_TAG_AcquisitionDeviceProcessingDescription	Acquisition Device Processing Description.
DCM_TAG_AcquisitionDeviceProcessingCode	Acquisition Device Processing Code.
DCM_TAG_CassetteOrientation	Cassette Orientation.
DCM_TAG_CassetteSize	Cassette Size.
DCM_TAG_ExposuresOnPlate	Exposures on Plate.
DCM_TAG_ExposureOnPlate	Exposures on Plate. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RelativeXrayExposure	Relative X-Ray Exposure. This

	tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ColumnAngulation	Column Angulation.
DCM_TAG_TomoLayerHeight	Tomo Layer Height.
DCM_TAG_TomoAngle	Tomo Angle.
DCM_TAG_TomoTime	Tomo Time.
DCM_TAG_TomoType	Tomo Type.
DCM_TAG_TomoClass	Tomo Class.
DCM_TAG_NumberOfTomosynthesisSourceImages	Number of Tomosynthesis Source Images.
DCM_TAG_TomoSourceImageNumber	Number of Tomosynthesis Source Images. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PositionerMotion	Positioner Motion.
DCM_TAG_PositionerType	Positioner Type.
DCM_TAG_PositionerPrimaryAngle	Positioner Primary Angle.
DCM_TAG_PositionerSecondaryAngle	Positioner Secondary Angle.
DCM_TAG_PositionerPrimaryAngleIncrement	Positioner Primary Angle Increment.
DCM_TAG_PositionerPrimaryAngleIncr	Positioner Primary Angle Increment. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PositionerSecondaryAngleIncrement	Positioner Secondary Angle Increment.
DCM_TAG_PositionerSecondaryAngleIncr	Positioner Secondary Angle Increment. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DetectorPrimaryAngle	Detector Primary Angle.
DCM_TAG_DetectorSecondaryAngle	Detector Secondary Angle.
DCM_TAG_ShutterShape	Shutter Shape.
DCM_TAG_ShutterLeftVerticalEdge	Shutter Left Vertical Edge.
DCM_TAG_ShutterRightVerticalEdge	Shutter Right Vertical Edge.
DCM_TAG_ShutterUpperHorizontalEdge	Shutter Upper Horizontal Edge.
DCM_TAG_ShutterLowerHorizontalEdge	Shutter Lower Horizontal Edge.
DCM_TAG_CenterOfCircularShutter	Center of Circular Shutter.
DCM_TAG_RadiusOfCircularShutter	Radius of Circular Shutter.
DCM_TAG_VerticesOfThePolygonalShutter	Vertices of the Polygonal Shutter.
DCM_TAG_PolygonalShutterVertices	Vertices of the Polygonal Shutter. This tag name has been deprecated and will be removed

	from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ShutterPresentationValue	Shutter Presentation Value.
DCM_TAG_ShutterOverlayGroup	Shutter Overlay Group.
DCM_TAG_ShutterPresentationColorCIELabValue	Shutter Presentation Color CIELab Value.
DCM_TAG_CollimatorShape	Collimator Shape.
DCM_TAG_CollimatorLeftVerticalEdge	Collimator Left Vertical Edge.
DCM_TAG_CollimatorRightVerticalEdge	Collimator Right Vertical Edge.
DCM_TAG_CollimatorUpperHorizontalEdge	Collimator Upper Horizontal Edge.
DCM_TAG_CollimatorLowerHorizontalEdge	Collimator Lower Horizontal Edge.
DCM_TAG_CenterOfCircularCollimator	Center of Circular Collimator.
DCM_TAG_RadiusOfCircularCollimator	Radius of Circular Collimator.
DCM_TAG_VerticesOfThePolygonalCollimator	Vertices of the Polygonal Collimator.
DCM_TAG_PolygonalCollimatorVertices	Vertices of the Polygonal Collimator. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AcquisitionTimeSynchronized	Acquisition Time Synchronized.
DCM_TAG_TimeSource	Time Source.
DCM_TAG_TimeDistributionProtocol	Time Distribution Protocol.
DCM_TAG_NTPSourceAddress	NTP Source Address.
DCM_TAG_PageNumberVector	Page Number Vector.
DCM_TAG_FrameLabelVector	Frame Label Vector.
DCM_TAG_FramePrimaryAngleVector	Frame Primary Angle Vector.
DCM_TAG_FrameSecondaryAngleVector	Frame Secondary Angle Vector.
DCM_TAG_SliceLocationVector	Slice Location Vector.
DCM_TAG_DisplayWindowLabelVector	Display Window Label Vector.
DCM_TAG_NominalScannedPixelSpacing	Nominal Scanned Pixel Spacing.
DCM_TAG_DigitizingDeviceTransportDirection	Digitizing Device Transport Direction.
DCM_TAG_RotationOfScannedFilm	Rotation of Scanned Film.
DCM_TAG_IvusAcquisition	IVUS Acquisition.
DCM_TAG_IvusPullbackRate	IVUS Pullback Rate.
DCM_TAG_IvusGatedRate	IVUS Gated Rate.
DCM_TAG_IvusPullbackStartFrameNumber	IVUS Pullback Start Frame Number.
DCM_TAG_IvusPullbackStopFrameNumber	IVUS Pullback Stop Frame Number.
DCM_TAG_LesionNumber	Lesion Number.
DCM_TAG_AcquisitionComments	Acquisition Comments. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OutputPower	Output Power.
DCM_TAG_TransducerData	Transducer Data.

DCM_TAG_FocusDepth	Focus Depth.
DCM_TAG_ProcessingFunction	Processing Function.
DCM_TAG_PreprocessingFunction	Processing Function. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PostprocessingFunction	Postprocessing Function. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MechanicalIndex	Mechanical Index.
DCM_TAG_BoneThermalIndex	Bone Thermal Index.
DCM_TAG_CranialThermalIndex	Cranial Thermal Index.
DCM_TAG_SoftTissueThermalIndex	Soft Tissue Thermal Index.
DCM_TAG_SoftTissueFocusThermalIndex	Soft Tissue-focus Thermal Index.
DCM_TAG_SoftTissueSurfaceThermalIndex	Soft Tissue-surface Thermal Index.
DCM_TAG_DynamicRange	Dynamic Range. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TotalGain	Total Gain. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DepthOfScanField	Depth of Scan Field.
DCM_TAG_PatientPosition	Patient Position.
DCM_TAG_ViewPosition	View Position.
DCM_TAG_ProjectionEponymousNameCodeSequence	Projection Eponymous Name Code Sequence.
DCM_TAG_ProjEponymousNameCodeSequence	Projection Eponymous Name Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ImageTransformationMatrix	Image Transformation Matrix. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageTranslationVector	Image Translation Vector. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Sensitivity	Sensitivity value.
DCM_TAG_SequenceOfUltrasoundRegions	Sequence of Ultrasound Regions.
DCM_TAG_RegionSpatialFormat	Region Spatial Format.
DCM_TAG_RegionDataType	Region Data Type.
DCM_TAG_RegionFlags	Region Flags.
DCM_TAG_RegionLocationMinX0	Region Location Min X0.
DCM_TAG_RegionLocationMinY0	Region Location Min Y0.
DCM_TAG_RegionLocationMaxX1	Region Location Max X1.
DCM_TAG_RegionLocationMaxY1	Region Location Max Y1.

DCM_TAG_ReferencePixelX0	Reference Pixel X0.
DCM_TAG_ReferencePixelY0	Reference Pixel Y0.
DCM_TAG_PhysicalUnitsXDirection	Physical Units X Direction.
DCM_TAG_PhysicalUnitsYDirection	Physical Units Y Direction.
DCM_TAG_ReferencePixelPhysicalValueX	Reference Pixel Physical Value X.
DCM_TAG_ReferencePixelPhysicalValueY	Reference Pixel Physical Value Y.
DCM_TAG_PhysicalDeltaX	Physical Delta X.
DCM_TAG_PhysicalDeltaY	Physical Delta Y.
DCM_TAG_TransducerFrequency	Transducer Frequency.
DCM_TAG_TransducerType	Transducer Type.
DCM_TAG_PulseRepetitionFrequency	Pulse Repetition Frequency.
DCM_TAG_DopplerCorrectionAngle	Doppler Correction Angle.
DCM_TAG_SteeringAngle	Steering Angle.
DCM_TAG_DopplerSampleVolumeXPositionRetired	Doppler Sample Volume X Position (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DopplerSampleVolumeXPosition	Doppler Sample Volume X Position.
DCM_TAG_DopplerSampleVolumeYPositionRetired	Doppler Sample Volume Y Position (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DopplerSampleVolumeYPosition	Doppler Sample Volume Y Position.
DCM_TAG_TMLinePositionX0Retired	TM-Line Position X0 (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TmLinePositionX0	TM-Line Position X0.
DCM_TAG_TMLinePositionY0Retired	TM-Line Position Y0 (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TmLinePositionY0	TM-Line Position Y0.
DCM_TAG_TMLinePositionX1Retired	TM-Line Position X1 (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TmLinePositionX1	TM-Line Position X1.
DCM_TAG_TMLinePositionY1Retired	TM-Line Position Y1 (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TmLinePositionY1	TM-Line Position Y1.
DCM_TAG_PixelComponentOrganization	Pixel Component Organization.
DCM_TAG_PixelComponentMask	Pixel Component Mask.
DCM_TAG_PixelComponentRangeStart	Pixel Component Range Start.
DCM_TAG_PixelComponentRangeStop	Pixel Component Range Stop.
DCM_TAG_PixelComponentPhysicalUnits	Pixel Component Physical Units.
DCM_TAG_PixelComponentDataType	Pixel Component Data Type.
DCM_TAG_NumberOfTableBreakPoints	Number of Table Break Points.

DCM_TAG_TableOfXBreakPoints	Table of X Break Points.
DCM_TAG_TableOfYBreakPoints	Table of Y Break Points.
DCM_TAG_NumberOfTableEntries	Number of Table Entries.
DCM_TAG_TableOfPixelValues	Table of Pixel Values.
DCM_TAG_TableOfParameterValues	Table of Parameter Values.
DCM_TAG_RWaveTimeVector	R Wave Time Vector.
DCM_TAG_R_Wave_Time_Vector	R Wave Time Vector. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DetectorConditionsNominalFlag	Detector Conditions Nominal Flag.
DCM_TAG_DetectorTemperature	Detector Temperature.
DCM_TAG_DetectorType	Detector Type.
DCM_TAG_DetectorConfiguration	Detector Configuration.
DCM_TAG_DetectorDescription	Detector Description.
DCM_TAG_DetectorMode	Detector Mode.
DCM_TAG_DetectorID	Detector ID.
DCM_TAG_DateOfLastDetectorCalibration	Date of Last Detector Calibration.
DCM_TAG_DetectorCalibrationDate	Date of Last Detector Calibration. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TimeOfLastDetectorCalibration	Time of Last Detector Calibration.
DCM_TAG_DetectorCalibrationTime	Time of Last Detector Calibration. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ExposuresOnDetectorSinceLastCalibration	Exposures on Detector Since Last Calibration.
DCM_TAG_ExposuresSinceCalibration	Exposures on Detector Since Last Calibration. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ExposuresOnDetectorSinceManufactured	Exposures on Detector Since Manufactured.
DCM_TAG_ExposuresSinceManufactured	Exposures on Detector Since Manufactured. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DetectorTimeSinceLastExposure	Detector Time Since Last Exposure.
DCM_TAG_DetectorTimeSinceExposure	Detector Time Since Last Exposure. This tag name has

	been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DetectorActiveTime	Detector Active Time.
DCM_TAG_DetectorActivationOffsetFromExposure	Detector Activation Offset From Exposure.
DCM_TAG_DetectorActivationOffset	Detector Activation Offset From Exposure. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DetectorBinning	Detector Binning.
DCM_TAG_DetectorElementPhysicalSize	Detector Element Physical Size.
DCM_TAG_DetectorElementSpacing	Detector Element Spacing.
DCM_TAG_DetectorActiveShape	Detector Active Shape.
DCM_TAG_DetectorActiveDimensions	Detector Active Dimension(s).
DCM_TAG_DetectorActiveDimension	Detector Active Dimension(s). This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DetectorActiveOrigin	Detector Active Origin.
DCM_TAG_DetectorManufacturerName	Detector Manufacturer Name.
DCM_TAG_DetectorManufacturerModelName	Detector Manufacturer's Model Name.
DCM_TAG_DetectorManufacturersModelName	Detector Manufacturer's Model Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FieldOfViewOrigin	Field of View Origin.
DCM_TAG_FovOrigin	Field of View Origin. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FieldOfViewRotation	Field of View Rotation.
DCM_TAG_FovRotation	Field of View Rotation. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FieldOfViewHorizontalFlip	Field of View Horizontal Flip.
DCM_TAG_FovHorizontalFlip	Field of View Horizontal Flip. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.

DCM_TAG_GridAbsorbingMaterial	Grid Absorbing Material.
DCM_TAG_GridSpacingMaterial	Grid Spacing Material.
DCM_TAG_GridThickness	Grid Thickness.
DCM_TAG_GridPitch	Grid Pitch.
DCM_TAG_GridAspectRatio	Grid Aspect Ratio.
DCM_TAG_GridPeriod	Grid Period.
DCM_TAG_GridFocalDistance	Grid Focal Distance.
DCM_TAG_FilterMaterial	Filter Material.
DCM_TAG_FilterThicknessMinimum	Filter Thickness Minimum.
DCM_TAG_FilterMinThickness	Filter Thickness Minimum. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FilterThicknessMaximum	Filter Thickness Maximum.
DCM_TAG_FilterMaxThickness	Filter Thickness Maximum. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FilterBeamPathLengthMinimum	Filter Beam Path Length Minimum.
DCM_TAG_FilterBeamPathLengthMaximum	Filter Beam Path Length Maximum.
DCM_TAG_ExposureControlMode	Exposure Control Mode.
DCM_TAG_ExposureControlModeDescription	Exposure Control Mode Description.
DCM_TAG_ExposureControlModeDesc	Exposure Control Mode Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ExposureStatus	Exposure Status.
DCM_TAG_PhototimerSetting	Phototimer Setting.
DCM_TAG_ExposureTimeInuS	Exposure Time in micro S.
DCM_TAG_ExposureTimeInMicroS	Exposure Time in micro S. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ExposureTimeIn_mS	Exposure Time in micro S. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_XRayTubeCurrentInuA	X-Ray Tube Current in micro A.
DCM_TAG_XRayTubeCurrentInMicroA	X-Ray Tube Current in micro A. This tag name has been deprecated and will be removed

	from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_XRayTubeCurrentInmA	X-Ray Tube Current in micro A. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ContentQualification	Content Qualification.
DCM_TAG_PulseSequenceName	Pulse Sequence Name.
DCM_TAG_MrImagingModifierSequence	MR Imaging Modifier Sequence.
DCM_TAG_EchoPulseSequence	Echo Pulse Sequence.
DCM_TAG_InversionRecovery	Inversion Recovery.
DCM_TAG_FlowCompensation	Flow Compensation.
DCM_TAG_MultipleSpinEcho	Multiple Spin Echo.
DCM_TAG_MultiPlanarExcitation	Multi-planar Excitation.
DCM_TAG_PhaseContrast	Phase Contrast.
DCM_TAG_TimeOfFlightContrast	Time of Flight Contrast.
DCM_TAG_Spoiling	Spoiling value.
DCM_TAG_SteadyStatePulseSequence	Steady State Pulse Sequence.
DCM_TAG_EchoPlanarPulseSequence	Echo Planar Pulse Sequence.
DCM_TAG_TagAngleFirstAxis	Tag Angle First Axis.
DCM_TAG_MagnetizationTransfer	Magnetization Transfer.
DCM_TAG_T2Preparation	T2 Preparation.
DCM_TAG_BloodSignalNulling	Blood Signal Nulling.
DCM_TAG_SaturationRecovery	Saturation Recovery.
DCM_TAG_SpectrallySelectedSuppression	Spectrally Selected Suppression.
DCM_TAG_SpectrallySelectedExcitation	Spectrally Selected Excitation.
DCM_TAG_SpatialPresaturation	Spatial Pre-saturation.
DCM_TAG_Tagging	Tagging value.
DCM_TAG_OverSamplingPhase	Oversampling Phase.
DCM_TAG_TagSpacingFirstDimension	Tag Spacing First Dimension.
DCM_TAG_GeometryOfKspaceTraversal	Geometry of k-Space Traversal.
DCM_TAG_SegmentedKspaceTraversal	Segmented k-Space Traversal.
DCM_TAG_RectilinearPhaseEncodeReordering	Rectilinear Phase Encode Reordering.
DCM_TAG_TagThickness	Tag Thickness.
DCM_TAG_PartialFourierDirection	Partial Fourier Direction.
DCM_TAG_CardiacSynchronizationTechnique	Cardiac Synchronization Technique.
DCM_TAG_CardiacSyncTechnique	Cardiac Synchronization Technique. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReceiveCoilManufacturerName	Receive Coil Manufacturer Name.
DCM_TAG_MrReceiveCoilSequence	MR Receive Coil Sequence.
DCM_TAG_ReceiveCoilType	Receive Coil Type.

DCM_TAG_QuadratureReceiveCoil	Quadrature Receive Coil.
DCM_TAG_MultiCoilDefinitionSequence	Multi-Coil Definition Sequence.
DCM_TAG_MultiCoilConfiguration	Multi-Coil Configuration.
DCM_TAG_MultiCoilElementName	Multi-Coil Element Name.
DCM_TAG_MultiCoilElementUsed	Multi-Coil Element Used.
DCM_TAG_MrTransmitCoilSequence	MR Transmit Coil Sequence.
DCM_TAG_TransmitCoilManufacturerName	Transmit Coil Manufacturer Name.
DCM_TAG_TransmitCoilType	Transmit Coil Type.
DCM_TAG_SpectralWidth	Spectral Width.
DCM_TAG_ChemicalShiftReference	Chemical Shift Reference.
DCM_TAG_VolumeLocalizationTechnique	Volume Localization Technique.
DCM_TAG_MrAcquisitionFrequencyEncodingSteps	MR Acquisition Frequency Encoding Steps.
DCM_TAG_MrAcqFrequencyEncodingSteps	MR Acquisition Frequency Encoding Steps. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_Decoupling	De-coupling value.
DCM_TAG_DecoupledNucleus	De-coupled Nucleus.
DCM_TAG_DecouplingFrequency	De-coupling Frequency.
DCM_TAG_DecouplingMethod	De-coupling Method.
DCM_TAG_DecouplingChemicalShiftReference	De-coupling Chemical Shift Reference.
DCM_TAG_KspaceFiltering	K-space Filtering.
DCM_TAG_TimeDomainFiltering	Time Domain Filtering.
DCM_TAG_NumberOfZeroFills	Number of Zero fills.
DCM_TAG_BaselineCorrection	Baseline Correction.
DCM_TAG_ParallelReductionFactorInPlane	Parallel Reduction Factor In-plane.
DCM_TAG_CardiacRRintervalSpecified	Cardiac R-R Interval Specified.
DCM_TAG_AcquisitionDuration	Acquisition Duration.
DCM_TAG_FrameAcquisitionDateTime	Frame Acquisition DateTime.
DCM_TAG_FrameAcqDatetime	Frame Acquisition DateTime. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DiffusionDirectionality	Diffusion Directionality.
DCM_TAG_DiffusionGradientDirectionSequence	Diffusion Gradient Direction Sequence.
DCM_TAG_ParallelAcquisition	Parallel Acquisition.
DCM_TAG_ParallelAcquisitionTechnique	Parallel Acquisition Technique.
DCM_TAG_ParallelAcqTechnique	Parallel Acquisition Technique. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.

DCM_TAG_InversionTimes	Inversion Times.
DCM_TAG_MetaboliteMapDescription	Metabolite Map Description.
DCM_TAG_MetaboliteMapDesc	Metabolite Map Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PartialFourier	Partial Fourier.
DCM_TAG_EffectiveEchoTime	Effective Echo Time.
DCM_TAG_MetaboliteMapCodeSequence	Metabolite Map Code Sequence.
DCM_TAG_ChemicalShiftSequence	Chemical Shift Sequence.
DCM_TAG_CardiacSignalSource	Cardiac Signal Source.
DCM_TAG_DiffusionBvalue	Diffusion b-value.
DCM_TAG_Diffusion_b_value	Diffusion b-value. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DiffusionGradientOrientation	Diffusion Gradient Orientation.
DCM_TAG_VelocityEncodingDirection	Velocity Encoding Direction.
DCM_TAG_VelocityEncodingMinimumValue	Velocity Encoding Minimum Value.
DCM_TAG_VelocityEncodingMinValue	Velocity Encoding Minimum Value. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_NumberOfKspaceTrajectories	Number of k-Space Trajectories.
DCM_TAG_CoverageOfKspace	Coverage of k-Space.
DCM_TAG_SpectroscopyAcquisitionPhaseRows	Spectroscopy Acquisition Phase Rows.
DCM_TAG_SpectroscopyAcqPhaseRows	Spectroscopy Acquisition Phase Rows. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ParallelReductionFactorInPlaneRetired	Parallel Reduction Factor In-plane (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TransmitterFrequency	Transmitter Frequency.
DCM_TAG_ResonantNucleus	Resonant Nucleus.
DCM_TAG_FrequencyCorrection	Frequency Correction.
DCM_TAG_MrSpectroscopyFovGeometrySequence	MR Spectroscopy FOV/Geometry Sequence.
DCM_TAG_SlabThickness	Slab Thickness.
DCM_TAG_SlabOrientation	Slab Orientation.
DCM_TAG_MidSlabPosition	Mid Slab Position.
DCM_TAG_MrSpatialSaturationSequence	MR Spatial Saturation Sequence.

DCM_TAG_MrTimingAndRelatedParametersSequence	MR Timing and Related Parameters Sequence.
DCM_TAG_MrTimingParameterSequence	MR Timing and Related Parameters Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MrEchoSequence	MR Echo Sequence.
DCM_TAG_MrModifierSequence	MR Modifier Sequence.
DCM_TAG_MrDiffusionSequence	MR Diffusion Sequence.
DCM_TAG_CardiacSynchronizationSequence	Cardiac Synchronization Sequence.
DCM_TAG_CardiacTriggerSequence	Cardiac Synchronization Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MrAveragesSequence	MR Averages Sequence.
DCM_TAG_MrFovGeometrySequence	MR FOV/Geometry Sequence.
DCM_TAG_VolumeLocalizationSequence	Volume Localization Sequence.
DCM_TAG_SpectroscopyAcquisitionDataColumns	Spectroscopy Acquisition Data Columns.
DCM_TAG_SpectroscopyAcqDataColumns	Spectroscopy Acquisition Data Columns. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DiffusionAnisotropyType	Diffusion Anisotropy Type.
DCM_TAG_FrameReferenceDatetime	Frame Reference DateTime. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MrMetaboliteMapSequence	MR Metabolite Map Sequence.
DCM_TAG_ParallelReductionFactorOutOfPlane	Parallel Reduction Factor out-of-plane.
DCM_TAG_SpectroscopyAcquisitionOutOfPlanePhaseSteps	Spectroscopy Acquisition Out-of-plane Phase Steps.
DCM_TAG_SpectroscopyAcqOutOfPlanePhaseStep	Spectroscopy Acquisition Out-of-plane Phase Steps. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BulkMotionStatus	Bulk Motion Status. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ParallelReductionFactorSecondInPlane	Parallel Reduction Factor Second In-plane.
DCM_TAG_CardiacBeatRejectionTechnique	Cardiac Beat Rejection

	Technique.
DCM_TAG_RespiratoryMotionCompensationTechnique	Respiratory Motion Compensation Technique.
DCM_TAG_RespiratoryMotionCompTechnique	Respiratory Motion Compensation Technique. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RespiratorySignalSource	Respiratory Signal Source.
DCM_TAG_BulkMotionCompensationTechnique	Bulk Motion Compensation Technique.
DCM_TAG_BulkMotionCompTechnique	Bulk Motion Compensation Technique. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BulkMotionSignalSource	Bulk Motion Signal Source.
DCM_TAG_ApplicableSafetyStandardAgency	Applicable Safety Standard Agency.
DCM_TAG_ApplicableSafetyStandardDescription	Applicable Safety Standard Description.
DCM_TAG_ApplicableSafetyStandardDesc	Applicable Safety Standard Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OperatingModeSequence	Operating Mode Sequence.
DCM_TAG_OperatingModeType	Operating Mode Type.
DCM_TAG_OperatingMode	Operating Mode.
DCM_TAG_SpecificAbsorptionRateDefinition	Specific Absorption Rate Definition.
DCM_TAG_GradientOutputType	Gradient Output Type.
DCM_TAG_SpecificAbsorptionRateValue	Specific Absorption Rate Value.
DCM_TAG_GradientOutput	Gradient Output.
DCM_TAG_FlowCompensationDirection	Flow Compensation Direction.
DCM_TAG_FlowCompDirection	Flow Compensation Direction. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TaggingDelay	Tagging Delay.
DCM_TAG_RespiratoryMotionCompensationTechniqueDescription	Respiratory Motion Compensation Technique Description.
DCM_TAG_RespiratorySignalSourceID	Respiratory Signal Source ID.
DCM_TAG_ChemicalShiftsMinimumIntegrationLimitInHz	Chemical Shifts Minimum Integration Limit in Hz. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.

DCM_TAG_ChemicalShiftsMaximumIntegrationLimitInHz	Chemical Shifts Maximum Integration Limit in Hz. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MrVelocityEncodingSequence	MR Velocity Encoding Sequence.
DCM_TAG_FirstOrderPhaseCorrection	First Order Phase Correction.
DCM_TAG_WaterReferencedPhaseCorrection	Water Referenced Phase Correction.
DCM_TAG_MrSpectroscopyAcquisitionType	MR Spectroscopy Acquisition Type.
DCM_TAG_MrSpectroscopyAcqType	MR Spectroscopy Acquisition Type. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RespiratoryCyclePosition	Respiratory Cycle Position.
DCM_TAG_VelocityEncodingMaximumValue	Velocity Encoding Maximum Value.
DCM_TAG_VelocityEncodingMaxValue	Velocity Encoding Maximum Value. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TagSpacingSecondDimension	Tag Spacing Second Dimension.
DCM_TAG_TagAngleSecondAxis	Tag Angle Second Axis.
DCM_TAG_FrameAcquisitionDuration	Frame Acquisition Duration.
DCM_TAG_FrameAcqDuration	Frame Acquisition Duration. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MrImageFrameTypeSequence	MR Image Frame Type Sequence.
DCM_TAG_MrSpectroscopyFrameTypeSequence	MR Spectroscopy Frame Type Sequence.
DCM_TAG_MrAcquisitionPhaseEncodingStepsInPlane	MR Acquisition Phase Encoding Steps in-plane.
DCM_TAG_MrAcqPhaseEncodingStepsInPlane	MR Acquisition Phase Encoding Steps in-plane. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MrAcquisitionPhaseEncodingStepsOutOfPlane	MR Acquisition Phase Encoding Steps out-of-plane.
DCM_TAG_MrAcqPhaseEncodingStepsOutOfPlane	MR Acquisition Phase Encoding Steps out-of-plane. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.

DCM_TAG_SpectroscopyAcquisitionPhaseColumns	Spectroscopy Acquisition Phase Columns.
DCM_TAG_SpectroscopyAcqPhaseColumns	Spectroscopy Acquisition Phase Columns. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CardiacCyclePosition	Cardiac Cycle Position.
DCM_TAG_SpecificAbsorptionRateSequence	Specific Absorption Rate Sequence.
DCM_TAG_RFEchoTrainLength	RF Echo Train Length.
DCM_TAG_GradientEchoTrainLength	Gradient Echo Train Length.
DCM_TAG_ChemicalShiftMinimumIntegrationLimitInppm	Chemical Shift Minimum Integration Limit in ppm.
DCM_TAG_ChemicalShiftsMinimumIntegrationLimitinppm	Chemical Shift Minimum Integration Limit in ppm. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ChemicalShiftMaximumIntegrationLimitInppm	Chemical Shift Maximum Integration Limit in ppm.
DCM_TAG_ChemicalShiftsMaximumIntegrationLimitinppm	Chemical Shift Maximum Integration Limit in ppm. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CTAcquisitionTypeSequence	CT Acquisition Type Sequence.
DCM_TAG_AcquisitionType	Acquisition Type.
DCM_TAG_TubeAngle	Tube Angle.
DCM_TAG_CTAcquisitionDetailsSequence	CT Acquisition Details Sequence.
DCM_TAG_RevolutionTime	Revolution Time.
DCM_TAG_SingleCollimationWidth	Single Collimation Width.
DCM_TAG_TotalCollimationWidth	Total Collimation Width.
DCM_TAG_CTTableDynamicsSequence	CT Table Dynamics Sequence.
DCM_TAG_TableSpeed	Table Speed.
DCM_TAG_TableFeedperRotation	Table Feed per Rotation.
DCM_TAG_SpiralPitchFactor	Spiral Pitch Factor.
DCM_TAG_CTGeometrySequence	CT Geometry Sequence.
DCM_TAG_DataCollectionCenterPatient	Data Collection Center (Patient).
DCM_TAG_CTReconstructionSequence	CT Reconstruction Sequence.
DCM_TAG_ReconstructionAlgorithm	Reconstruction Algorithm.
DCM_TAG_ConvolutionKernelGroup	Convolution Kernel Group.
DCM_TAG_ReconstructionFieldofView	Reconstruction Field of View.
DCM_TAG_ReconstructionTargetCenterPatient	Reconstruction Target Center (Patient).
DCM_TAG_ReconstructionAngle	Reconstruction Angle.
DCM_TAG_ImageFilter	Image Filter.

DCM_TAG_CTExposureSequence	CT Exposure Sequence.
DCM_TAG_ReconstructionPixelSpacing	Reconstruction Pixel Spacing.
DCM_TAG_ExposureModulationType	Exposure Modulation Type.
DCM_TAG_EstimatedDoseSaving	Estimated Dose Saving.
DCM_TAG_CTXRayDetailsSequence	CT X-Ray Details Sequence.
DCM_TAG_CTX_rayDetailsSequence	CT X-Ray Details Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CTPositionSequence	CT Position Sequence.
DCM_TAG_TablePosition	Table Position.
DCM_TAG_ExposureTimeinms	Exposure Time in ms.
DCM_TAG_CTImageFrameTypeSequence	CT Image Frame Type Sequence.
DCM_TAG_XRayTubeCurrentInMilliA	X-Ray Tube Current in mA.
DCM_TAG_X_RayTubeCurrentinmA	X-Ray Tube Current in mA. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ExposureinmAs	Exposure in mAs.
DCM_TAG_ExposureInMilliAs	Exposure in mAs. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ConstantVolumeFlag	Constant Volume Flag.
DCM_TAG_FluoroscopyFlag	Fluoroscopy Flag.
DCM_TAG_DistanceSourcetoDataCollectionCenter	Distance Source to Data Collection Center.
DCM_TAG_ContrastBolusAgentNumber	Contrast/Bolus Agent Number.
DCM_TAG_ContrastBolusIngredientCodeSequence	Contrast/Bolus Ingredient Code Sequence.
DCM_TAG_ContrastAdministrationProfileSequence	Contrast Administration Profile Sequence.
DCM_TAG_ContrastBolusUsageSequence	Contrast/Bolus Usage Sequence.
DCM_TAG_ContrastBolusAgentAdministered	Contrast/Bolus Agent Administered.
DCM_TAG_ContrastBolusAgentDetected	Contrast/Bolus Agent Detected.
DCM_TAG_ContrastBolusAgentPhase	Contrast/Bolus Agent Phase.
DCM_TAG_CTDIvol	The CTDIvol.
DCM_TAG_CTDIPhantomTypeCodeSequence	CTDI Phantom Type Code Sequence.
DCM_TAG_CalciumScoringMassFactorPatient	Calcium Scoring Mass Factor Patient.
DCM_TAG_CalciumScoringMassFactorDevice	Calcium Scoring Mass Factor Device.
DCM_TAG_EnergyWeightingFactor	Energy Weighting Factor.
DCM_TAG_CTAdditionalXRaySourceSequence	CT Additional X-Ray Source Sequence.

DCM_TAG_ProjectionPixelCalibrationSequence	Projection Pixel Calibration Sequence.
DCM_TAG_DistanceSourcetoIsocenter	Distance Source to Isocenter.
DCM_TAG_DistanceObjecttoTableTop	Distance Object to Table Top.
DCM_TAG_ObjectPixelSpacinginCenterofBeam	Object Pixel Spacing in Center of Beam.
DCM_TAG_PositionerPositionSequence	Positioner Position Sequence.
DCM_TAG_TablePositionSequence	Table Position Sequence.
DCM_TAG_CollimatorShapeSequence	Collimator Shape Sequence.
DCM_TAG_XAXRFFrameCharacteristicsSequence	XA/XRF Frame Characteristics Sequence.
DCM_TAG_XA_XRFFrameCharacteristicsSequence	XA/XRF Frame Characteristics Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FrameAcquisitionSequence	Frame Acquisition Sequence.
DCM_TAG_XRayReceptorType	X-Ray Receptor Type.
DCM_TAG_AcquisitionProtocolName	Acquisition Protocol Name.
DCM_TAG_AcquisitionProtocolDescription	Acquisition Protocol Description.
DCM_TAG_ContrastBolusIngredientOpaque	Contrast/Bolus Ingredient Opaque.
DCM_TAG_DistanceReceptorPlanetoDetectorHousing	Distance Receptor Plane to Detector Housing.
DCM_TAG_IntensifierActiveShape	Intensifier Active Shape.
DCM_TAG_IntensifierActiveDimensions	Intensifier Active Dimension(s).
DCM_TAG_PhysicalDetectorSize	Physical Detector Size.
DCM_TAG_PositionofIsocenterProjection	Position of Isocenter Projection.
DCM_TAG_FieldofViewSequence	Field of View Sequence.
DCM_TAG_FieldofViewDescription	Field of View Description.
DCM_TAG_ExposureControlSensingRegionsSequence	Exposure Control Sensing Regions Sequence.
DCM_TAG_ExposureControlSensingRegionShape	Exposure Control Sensing Region Shape.
DCM_TAG_ExposureControlSensingRegionLeftVerticalEdge	Exposure Control Sensing Region Left Vertical Edge.
DCM_TAG_ExposureControlSensingRegionRightVerticalEdge	Exposure Control Sensing Region Right Vertical Edge.
DCM_TAG_ExposureControlSensingRegionUpperHorizontalEdge	Exposure Control Sensing Region Upper Horizontal Edge.
DCM_TAG_ExposureControlSensingRegionUpperHorizontal	Exposure Control Sensing Region Upper Horizontal Edge. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ExposureControlSensingRegionLowerHorizontalEdge	Exposure Control Sensing Region Lower Horizontal Edge.
DCM_TAG_ExposureControlSensingRegionLowerHorizontal	Exposure Control Sensing Region Lower Horizontal Edge. This tag name has been deprecated and will be removed from the public API in a future release. Please

	use the tag with the same value defined in the previous line.
DCM_TAG_CenterofCircularExposureControlSensingRegion	Center of Circular Exposure Control Sensing Region.
DCM_TAG_RadiusofCircularExposureControlSensingRegion	Radius of Circular Exposure Control Sensing Region.
DCM_TAG_VerticesOfThePolygonalExposureControlSensingRegion	Vertices of the Polygonal Exposure Control Sensing Region.
DCM_TAG_VerticesofthePolygonalExposureControlSensing	Vertices of the Polygonal Exposure Control Sensing Region. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RET	RET. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ColumnAngulationPatient	Column Angulation (Patient).
DCM_TAG_BeamAngle	Beam Angle.
DCM_TAG_FrameDetectorParametersSequence	Frame Detector Parameters Sequence.
DCM_TAG_CalculatedAnatomyThickness	Calculated Anatomy Thickness.
DCM_TAG_CalibrationSequence	Calibration Sequence.
DCM_TAG_ObjectThicknessSequence	Object Thickness Sequence.
DCM_TAG_PlaneIdentification	Plane Identification.
DCM_TAG_FieldofViewDimensionsInFloat	Field of View Dimension(s) in Float.
DCM_TAG_IsocenterReferenceSystemSequence	Isocenter Reference System Sequence.
DCM_TAG_PositionerIsocenterPrimaryAngle	Positioner Isocenter Primary Angle.
DCM_TAG_PositionerIsocenterSecondaryAngle	Positioner Isocenter Secondary Angle.
DCM_TAG_PositionerIsocenterDetectorRotationAngle	Positioner Isocenter Detector Rotation Angle.
DCM_TAG_TableXPositiontoIsocenter	Table X Position to Isocenter.
DCM_TAG_TableYPositiontoIsocenter	Table Y Position to Isocenter.
DCM_TAG_TableZPositiontoIsocenter	Table Z Position to Isocenter.
DCM_TAG_TableHorizontalRotationAngle	Table Horizontal Rotation Angle.
DCM_TAG_TableHeadTiltAngle	Table Head Tilt Angle.
DCM_TAG_TableCradleTiltAngle	Table Cradle Tilt Angle.
DCM_TAG_FrameDisplayShutterSequence	Frame Display Shutter Sequence.
DCM_TAG_AcquiredImageAreaDoseProduct	Acquired Image Area Dose Product.
DCM_TAG_CarmPositionerTabletopRelationship	C-arm Positioner Tabletop Relationship.
DCM_TAG_XRayGeometrySequence	X-Ray Geometry Sequence.
DCM_TAG_IrradiationEventIdentificationSequence	Irradiation Event Identification Sequence.
DCM_TAG_XRay3DFrameTypeSequence	X-Ray 3D Frame Type Sequence.
DCM_TAG_ContributingSourcesSequence	Contributing Sources Sequence.

DCM_TAG_XRay3DAcquisitionSequence	X-Ray 3D Acquisition Sequence.
DCM_TAG_PrimaryPositionerScanArc	Primary Positioner Scan Arc.
DCM_TAG_SecondaryPositionerScanArc	Secondary Positioner Scan Arc.
DCM_TAG_PrimaryPositionerScanStartAngle	Primary Positioner Scan Start Angle.
DCM_TAG_SecondaryPositionerScanStartAngle	Secondary Positioner Scan Start Angle.
DCM_TAG_PrimaryPositionerIncrement	Primary Positioner Increment.
DCM_TAG_SecondaryPositionerIncrement	Secondary Positioner Increment.
DCM_TAG_StartAcquisitionDateTime	Start Acquisition DateTime.
DCM_TAG_EndAcquisitionDateTime	End Acquisition DateTime.
DCM_TAG_ApplicationName	Application Name.
DCM_TAG_ApplicationVersion	Application Version.
DCM_TAG_ApplicationManufacturer	Application Manufacturer.
DCM_TAG_AlgorithmType	Algorithm Type.
DCM_TAG_AlgorithmDescription	Algorithm Description.
DCM_TAG_XRay3DReconstructionSequence	X-Ray 3D Reconstruction Sequence.
DCM_TAG_ReconstructionDescription	Reconstruction Description.
DCM_TAG_PerProjectionAcquisitionSequence	Per Projection Acquisition Sequence.
DCM_TAG_DiffusionBMatrixSequence	Diffusion b-matrix Sequence.
DCM_TAG_Diffusion_b_matrixSequence	Diffusion b-matrix Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DiffusionBValueXX	Diffusion b-value XX.
DCM_TAG_Diffusion_b_valueXX	Diffusion b-value XX. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DiffusionBValueXY	Diffusion b-value XY.
DCM_TAG_Diffusion_b_valueXY	Diffusion b-value XY. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DiffusionBValueXZ	Diffusion b-value XZ.
DCM_TAG_Diffusion_b_valueXZ	Diffusion b-value XZ. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DiffusionBValueYY	Diffusion b-value YY.
DCM_TAG_Diffusion_b_valueYY	Diffusion b-value YY. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.

DCM_TAG_DiffusionBValueYZ	Diffusion b-value YZ.
DCM_TAG_Diffusion_b_valueYZ	Diffusion b-value YZ. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DiffusionBValueZZ	Diffusion b-value ZZ.
DCM_TAG_Diffusion_b_valueZZ	Diffusion b-value ZZ. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DecayCorrectionDateTime	Decay Correction DateTime.
DCM_TAG_StartDensityThreshold	Start Density Threshold.
DCM_TAG_StartRelativeDensityDifferenceThreshold	Start Relative Density Difference Threshold.
DCM_TAG_StartCardiacTriggerCountThreshold	Start Cardiac Trigger Count Threshold.
DCM_TAG_StartRespiratoryTriggerCountThreshold	Start Respiratory Trigger Count Threshold.
DCM_TAG_TerminationCountsThreshold	Termination Counts Threshold.
DCM_TAG_TerminationDensityThreshold	Termination Density Threshold.
DCM_TAG_TerminationRelativeDensityThreshold	Termination Relative Density Threshold.
DCM_TAG_TerminationTimeThreshold	Termination Time Threshold.
DCM_TAG_TerminationCardiacTriggerCountThreshold	Termination Cardiac Trigger Count Threshold.
DCM_TAG_TerminationRespiratoryTriggerCountThreshold	Termination Respiratory Trigger Count Threshold.
DCM_TAG_DetectorGeometry	Detector Geometry.
DCM_TAG_TransverseDetectorSeparation	Transverse Detector Separation.
DCM_TAG_AxialDetectorDimension	Axial Detector Dimension.
DCM_TAG_RadiopharmaceuticalAgentNumber	Radiopharmaceutical Agent Number.
DCM_TAG_PETFrameAcquisitionSequence	PET Frame Acquisition Sequence.
DCM_TAG_PETDetectorMotionDetailsSequence	PET Detector Motion Details Sequence.
DCM_TAG_PETTableDynamicsSequence	PET Table Dynamics Sequence.
DCM_TAG_PETPositionSequence	PET Position Sequence.
DCM_TAG_PETFrameCorrectionFactorsSequence	PET Frame Correction Factors Sequence.
DCM_TAG_RadiopharmaceuticalUsageSequence	Radiopharmaceutical Usage Sequence.
DCM_TAG_AttenuationCorrectionSource	Attenuation Correction Source.
DCM_TAG_NumberOfIterations	Number of Iterations.
DCM_TAG_NumberOfSubsets	Number of Subsets.
DCM_TAG_PETReconstructionSequence	PET Reconstruction Sequence.
DCM_TAG_PETFrameTypeSequence	PET Frame Type Sequence.
DCM_TAG_TimeOfFlightInformationUsed	Time of Flight Information Used.
DCM_TAG_ReconstructionType	Reconstruction Type.
DCM_TAG_DecayCorrected	Decay Corrected.
DCM_TAG_AttenuationCorrected	Attenuation Corrected.

DCM_TAG_ScatterCorrected	Scatter Corrected.
DCM_TAG_DeadTimeCorrected	Dead Time Corrected.
DCM_TAG_GantryMotionCorrected	Gantry Motion Corrected.
DCM_TAG_PatientMotionCorrected	Patient Motion Corrected.
DCM_TAG_CountLossNormalizationCorrected	Count Loss Normalization Corrected.
DCM_TAG_RandomsCorrected	Randoms Corrected.
DCM_TAG_NonUniformRadialSamplingCorrected	Non-uniform Radial Sampling Corrected.
DCM_TAG_SensitivityCalibrated	Sensitivity Calibrated.
DCM_TAG_DetectorNormalizationCorrection	Detector Normalization Correction.
DCM_TAG_IterativeReconstructionMethod	Iterative Reconstruction Method.
DCM_TAG_AttenuationCorrectionTemporalRelationship	Attenuation Correction Temporal Relationship.
DCM_TAG_PatientPhysiologicalStateSequence	Patient Physiological State Sequence.
DCM_TAG_PatientPhysiologicalStateCodeSequence	Patient Physiological State Code Sequence.
DCM_TAG_DepthsOfFocus	Depth(s) of Focus.
DCM_TAG_ExcludedIntervalsSequence	Excluded Intervals Sequence.
DCM_TAG_ExclusionStartDatetime	Exclusion Start Datetime.
DCM_TAG_ExclusionDuration	Exclusion Duration.
DCM_TAG_USImageDescriptionSequence	US Image Description Sequence.
DCM_TAG_ImageDataTypeSequence	Image Data Type Sequence.
DCM_TAG_DataType	Data Type.
DCM_TAG_TransducerScanPatternCodeSequence	Transducer Scan Pattern Code Sequence.
DCM_TAG_AliasedDataType	Aliased Data Type.
DCM_TAG_PositionMeasuringDeviceUsed	Position Measuring Device Used.
DCM_TAG_TransducerGeometryCodeSequence	Transducer Geometry Code Sequence.
DCM_TAG_TransducerBeamSteeringCodeSequence	Transducer Beam Steering Code Sequence.
DCM_TAG_TransducerApplicationCodeSequence	Transducer Application Code Sequence.
DCM_TAG_ContributingEquipmentSequence	Contributing Equipment Sequence.
DCM_TAG_ContributionDateTime	Contribution Date Time.
DCM_TAG_ContributionDescription	Contribution Description.
DCM_TAG_ContributionDesc	Contribution Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_Group0020Length	Group 0020 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyInstanceUID	Study Instance UID.
DCM_TAG_SeriesInstanceUID	Series Instance UID.
DCM_TAG_StudyID	Study Identifier.

DCM_TAG_SeriesNumber	Series Number.
DCM_TAG_AcquisitionNumber	Acquisition Number.
DCM_TAG_InstanceNumber	Instance Number.
DCM_TAG_IsotopeNumber	Isotope Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PhaseNumber	Phase Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_IntervalNumber	Interval Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TimeSlotNumber	Time Slot Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AngleNumber	Angle Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ItemNumber	Item Number.
DCM_TAG_PatientOrientation	Patient Orientation.
DCM_TAG_OverlayNumber	Overlay Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveNumber	Curve Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LUTNumber	LUT Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LookupTableNumber	Lookup Table Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImagePosition	Image Position. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImagePositionPatient	Image Position (Patient).
DCM_TAG_ImageOrientation	Image Orientation. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageOrientationPatient	Image Orientation (Patient).
DCM_TAG_Location	Location. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_FrameOfReferenceUID	Frame of Reference UID.
DCM_TAG_Laterality	The Laterality.
DCM_TAG_ImageLaterality	Image Laterality.
DCM_TAG_ImageGeometryType	Image Geometry Type. This tag

	is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MaskingImage	Masking Image. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TemporalPositionIdentifier	Temporal Position Identifier.
DCM_TAG_NumberOfTemporalPositions	Number of Temporal Positions.
DCM_TAG_TemporalResolution	Temporal Resolution.
DCM_TAG_SynchronizationFrameOfReferenceUID	Synchronization Frame of Reference UID.
DCM_TAG_SyncFrameOfRefUID	Synchronization Frame of Reference UID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SOPInstanceUIDOfConcatenationSource	SOP Instance UID of Concatenation Source.
DCM_TAG_SeriesInStudy	Series in Study. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AcquisitionsInSeries	Acquisitions in Series. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImagesInAcquisition	Images in Acquisition.
DCM_TAG_ImagesInSeries	Images in Series. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AcquisitionsInStudy	Acquisitions in Study. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImagesInStudy	Images in Study. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Reference	Reference. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PositionReferenceIndicator	Position Reference Indicator.
DCM_TAG_SliceLocation	Slice Location.
DCM_TAG_OtherStudyNumbers	Other Study Numbers. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfPatientRelatedStudies	Number of Patient Related Studies.
DCM_TAG_NumberOfPatientRelatedSeries	Number of Patient Related Series.
DCM_TAG_NumberOfPatientRelatedInstances	Number of Patient Related Instances.
DCM_TAG_NumberOfStudyRelatedSeries	Number of Study Related Series.

DCM_TAG_NumberOfStudyRelatedInstances	Number of Study Related Instances.
DCM_TAG_NumberOfSeriesRelatedInstances	Number of Series Related Instances.
DCM_TAG_NumberOfSeriesRelatedInstance	Number of Series Related Instances. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SourceImageIDs	Source Image IDs. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ModifyingDeviceID	Modifying Device ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ModifiedImageID	Modified Image ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ModifiedImageDate	Modified Image Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ModifyingDeviceManufacturer	Modifying Device Manufacturer. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ModifiedImageTime	Modified Image Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ModifiedImageDescription	Modified Image Description. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageComments	Image Comments.
DCM_TAG_OriginalImageIdentification	Original Image Identification. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OriginalImageIdentificationNomenclature	Original Image Identification Nomenclature. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StackID	Stack Identifier.
DCM_TAG_InStackPositionNumber	In-Stack Position Number.
DCM_TAG_FrameAnatomySequence	Frame Anatomy Sequence.
DCM_TAG_FrameLaterality	Frame Laterality.
DCM_TAG_FrameContentSequence	Frame Content Sequence.
DCM_TAG_PlanePositionSequence	Plane Position Sequence.
DCM_TAG_PlaneOrientationSequence	Plane Orientation Sequence.
DCM_TAG_TemporalPositionIndex	Temporal Position Index.
DCM_TAG_NominalCardiacTriggerDelayTime	Nominal Cardiac Trigger Delay Time.

DCM_TAG_TriggerDelayTime	Nominal Cardiac Trigger Delay Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FrameAcquisitionNumber	Frame Acquisition Number.
DCM_TAG_FrameAcqNumber	Frame Acquisition Number. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DimensionIndexValues	Dimension Index Values.
DCM_TAG_DimensionIndexValue	Dimension Index Values. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FrameComments	Frame Comments.
DCM_TAG_ConcatenationUID	Concatenation UID.
DCM_TAG_InConcatenationNumber	In-concatenation Number.
DCM_TAG_InConcatenationTotalNumber	In-concatenation Total Number.
DCM_TAG_DimensionOrganizationUID	Dimension Organization UID.
DCM_TAG_DimensionIndexPointer	Dimension Index Pointer.
DCM_TAG_FunctionalGroupPointer	Functional Group Pointer.
DCM_TAG_DimensionIndexPrivateCreator	Dimension Index Private Creator.
DCM_TAG_DimensionOrganizationSequence	Dimension Organization Sequence.
DCM_TAG_DimensionIndexSequence	Dimension Index Sequence.
DCM_TAG_ConcatenationFrameOffsetNumber	Concatenation Frame Offset Number.
DCM_TAG_FunctionalGroupPrivateCreator	Functional Group Private Creator.
DCM_TAG_NominalPercentageOfCardiacPhase	Nominal Percentage of Cardiac Phase.
DCM_TAG_NominalPercentageOfRespiratoryPhase	Nominal Percentage of Respiratory Phase.
DCM_TAG_StartingRespiratoryAmplitude	Starting Respiratory Amplitude.
DCM_TAG_StartingRespiratoryPhase	Starting Respiratory Phase.
DCM_TAG_EndingRespiratoryAmplitude	Ending Respiratory Amplitude.
DCM_TAG_EndingRespiratoryPhase	Ending Respiratory Phase.
DCM_TAG_RespiratoryTriggerType	Respiratory Trigger Type.
DCM_TAG_RRIntervalTimeNominal	R - R Interval Time Nominal.
DCM_TAG_RRIntervalTimeMeasured	R - R Interval Time Nominal. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ActualCardiacTriggerDelayTime	Actual Cardiac Trigger Delay Time.
DCM_TAG_RespiratorySynchronizationSequence	Respiratory Synchronization Sequence.

DCM_TAG_RespiratoryTriggerSequence	Respiratory Synchronization Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RespiratoryIntervalTime	Respiratory Interval Time.
DCM_TAG_NominalRespiratoryTriggerDelayTime	Nominal Respiratory Trigger Delay Time.
DCM_TAG_RespiratoryTriggerDelayTime	Nominal Respiratory Trigger Delay Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RespiratoryTriggerDelayThreshold	Respiratory Trigger Delay Threshold.
DCM_TAG_ActualRespiratoryTriggerDelayTime	Actual Respiratory Trigger Delay Time.
DCM_TAG_ImagePositionVolume	Image Position (Volume).
DCM_TAG_ImageOrientationVolume	Image Orientation (Volume).
DCM_TAG_UltrasoundAcquisitionGeometry	Ultrasound Acquisition Geometry.
DCM_TAG_ApexPosition	Apex Position.
DCM_TAG_VolumeToTransducerMappingMatrix	Volume to Transducer Mapping Matrix.
DCM_TAG_VolumeToTableMappingMatrix	Volume to Table Mapping Matrix.
DCM_TAG_PatientFrameOfReferenceSource	Patient Frame of Reference Source.
DCM_TAG_TemporalPositionTimeOffset	Temporal Position Time Offset.
DCM_TAG_PlanePositionVolumeSequence	Plane Position (Volume) Sequence.
DCM_TAG_PlaneOrientationVolumeSequence	Plane Orientation (Volume) Sequence.
DCM_TAG_TemporalPositionSequence	Temporal Position Sequence.
DCM_TAG_DimensionOrganizationType	Dimension Organization Type.
DCM_TAG_VolumeFrameOfReferenceUID	Volume Frame of Reference UID.
DCM_TAG_TableFrameOfReferenceUID	Table Frame of Reference UID.
DCM_TAG_DimensionDescriptionLabel	Dimension Description Label.
DCM_TAG_PatientOrientationinFrameSequence	Patient Orientation in Frame Sequence.
DCM_TAG_FrameLabel	Frame Label.
DCM_TAG_AcquisitionIndex	Acquisition Index.
DCM_TAG_ContributingSOPInstancesReferenceSequence	Contributing SOP Instances Reference Sequence.
DCM_TAG_ReconstructionIndex	Reconstruction Index.
DCM_TAG_Group0022Length	Group 0022 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LightPathFilterPassThroughWavelength	Light Path Filter Pass-Through Wavelength.
DCM_TAG_LightPathFilterPassBand	Light Path Filter Pass Band.
DCM_TAG_ImagePathFilterPassThroughWavelength	Image Path Filter Pass-Through Wavelength.

DCM_TAG_ImagePathFilterPassBand	Image Path Filter Pass Band.
DCM_TAG_PatientEyeMovementCommanded	Patient Eye Movement Commanded.
DCM_TAG_PatientEyeMovementCommandCodeSequence	Patient Eye Movement Command Code Sequence.
DCM_TAG_SphericalLensPower	Spherical Lens Power.
DCM_TAG_CylinderLensPower	Cylinder Lens Power.
DCM_TAG_CylinderAxis	Cylinder Axis.
DCM_TAG_EmmetropicMagnification	Emmetropic Magnification.
DCM_TAG_IntraOcularPressure	Intra Ocular Pressure.
DCM_TAG_HorizontalFieldofView	Horizontal Field of View.
DCM_TAG_PupilDilated	Pupil Dilated.
DCM_TAG_DegreeofDilation	Degree of Dilation.
DCM_TAG_StereoBaselineAngle	Stereo Baseline Angle.
DCM_TAG_StereoBaselineDisplacement	Stereo Baseline Displacement.
DCM_TAG_StereoHorizontalPixelOffset	Stereo Horizontal Pixel Offset.
DCM_TAG_StereoVerticalPixelOffset	Stereo Vertical Pixel Offset.
DCM_TAG_StereoRotation	Stereo Rotation.
DCM_TAG_AcquisitionDeviceTypeCodeSequence	Acquisition Device Type Code Sequence.
DCM_TAG_IlluminationTypeCodeSequence	Illumination Type Code Sequence.
DCM_TAG_LightPathFilterTypeStackCodeSequence	Light Path Filter Type Stack Code Sequence.
DCM_TAG_ImagePathFilterTypeStackCodeSequence	Image Path Filter Type Stack Code Sequence.
DCM_TAG_LensesCodeSequence	Lenses Code Sequence.
DCM_TAG_ChannelDescriptionCodeSequence	Channel Description Code Sequence.
DCM_TAG_RefractiveStateSequence	Refractive State Sequence.
DCM_TAG_MydriaticAgentCodeSequence	Mydriatic Agent Code Sequence.
DCM_TAG_RelativeImagePositionCodeSequence	Relative Image Position Code Sequence.
DCM_TAG_StereoPairsSequence	Stereo Pairs Sequence.
DCM_TAG_LeftImageSequence	Left Image Sequence.
DCM_TAG_RightImageSequence	Right Image Sequence.
DCM_TAG_AxialLengthOfTheEye	Axial Length of the Eye.
DCM_TAG_OphthalmicFrameLocationSequence	Ophthalmic Frame Location Sequence.
DCM_TAG_ReferenceCoordinates	Reference Coordinates.
DCM_TAG_DepthSpatialResolution	Depth Spatial Resolution.
DCM_TAG_MaximumDepthDistortion	Maximum Depth Distortion.
DCM_TAG_AlongScanSpatialResolution	Along-scan Spatial Resolution.
DCM_TAG_MaximumAlongScanDistortion	Maximum Along-scan Distortion.
DCM_TAG_OphthalmicImageOrientation	Ophthalmic Image Orientation.
DCM_TAG_DepthOfTransverseImage	Depth of Transverse Image.
DCM_TAG_MydriaticAgentConcentrationUnitsSequence	Mydriatic Agent Concentration Units Sequence.
DCM_TAG_AcrossScanSpatialResolution	Across-scan Spatial Resolution.
DCM_TAG_MaximumAcrossScanDistortion	Maximum Across-scan Distortion.
DCM_TAG_MydriaticAgentConcentration	Mydriatic Agent Concentration.

DCM_TAG_IlluminationWaveLength	Illumination Wave Length.
DCM_TAG_IlluminationPower	Illumination Power.
DCM_TAG_IlluminationBandwidth	Illumination Bandwidth.
DCM_TAG_MydriaticAgentSequence	Mydriatic Agent Sequence.
DCM_TAG_Group0028Length	Group 0028 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SamplesPerPixel	Samples per Pixel.
DCM_TAG_SamplesperPixelUsed	Samples per Pixel Used.
DCM_TAG_PhotometricInterpretation	Photometric Interpretation.
DCM_TAG_ImageDimensions	Image Dimensions. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PlanarConfiguration	Planar Configuration.
DCM_TAG_NumberOfFrames	Number of Frames.
DCM_TAG_FrameIncrementPointer	Frame Increment Pointer.
DCM_TAG_FrameDimensionPointer	Frame Dimension Pointer.
DCM_TAG_Rows	Rows. Height of the DICOM image.
DCM_TAG_Columns	Columns. Width of the DICOM image.
DCM_TAG_Planes	Planes. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_UltrasoundColorDataPresent	Ultrasound Color Data Present.
DCM_TAG_PixelSpacing	Pixel Spacing.
DCM_TAG_ZoomFactor	Zoom Factor.
DCM_TAG_ZoomCenter	Zoom Center.
DCM_TAG_PixelAspectRatio	Pixel Aspect Ratio.
DCM_TAG_ImageFormat	Image Format. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ManipulatedImage	Manipulated Image. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CorrectedImage	Corrected Image.
DCM_TAG_CompressionRecognitionCode	Compression Recognition Code. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CompressionCode	Compression Code. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CompressionOriginator	Compression Originator. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CompressionLabel	Compression Label. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.

DCM_TAG_CompressionDescription	Compression Description. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CompressionSequence	Compression Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CompressionStepPointers	Compression Step Pointers. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RepeatInterval	Repeat Interval. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_BitsGrouped	Bits Grouped. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PerimeterTable	Perimeter Table. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PerimeterValue	Perimeter Value. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PredictorRows	Predictor Rows. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PredictorColumns	Predictor Columns. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PredictorConstants	Predictor Constants. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_BlockedPixels	Blocked Pixels. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_BlockRows	Block Rows. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_BlockColumns	Block Columns. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RowOverlap	Row Overlap. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ColumnOverlap	Column Overlap. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_BitsAllocated	Bits Allocated.
DCM_TAG_BitsStored	Bits Stored.

DCM_TAG_HighBit	Image High Bit.
DCM_TAG_PixelRepresentation	Pixel Representation.
DCM_TAG_SmallestValidPixelValue	Smallest Valid Pixel Value. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LargestValidPixelValue	Largest Valid Pixel Value. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SmallestImagePixelValue	Smallest Image Pixel Value.
DCM_TAG_LargestImagePixelValue	Largest Image Pixel Value.
DCM_TAG_SmallestPixelValueInSeries	Smallest Pixel Value in Series.
DCM_TAG_LargestPixelValueInSeries	Largest Pixel Value in Series.
DCM_TAG_SmallestImagePixelValueInPlane	Smallest Image Pixel Value in Plane. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LargestImagePixelValueInPlane	Largest Image Pixel Value in Plane. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PixelPaddingValue	Pixel Padding Value.
DCM_TAG_PixelPaddingRangeLimit	Pixel Padding Range Limit.
DCM_TAG_ImageLocation	Image Location. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_QualityControlImage	Quality Control Image.
DCM_TAG_BurnedInAnnotation	Burned In Annotation.
DCM_TAG_TransformLabel	Transform Label. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TransformVersionNumber	Transform Version Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfTransformSteps	Number of Transform Steps. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SequenceOfCompressedData	Sequence of Compressed Data. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DetailsOfCoefficients	Details of Coefficients. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RowsForNthOrderCoefficients	Rows For Nth Order Coefficients. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ColumnsForNthOrderCoefficients	Columns For Nth Order Coefficients. This tag is marked as retired in DICOM specification. See DICOM specification for

	alternatives.
DCM_TAG_CoefficientCoding	Coefficient Coding. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CoefficientCodingPointers	Coefficient Coding Pointers. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DCTLabel	DCT Label. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DataBlockDescription	Data Block Description. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DataBlock	Data Block. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NormalizationFactorFormat	Normalization Factor Format. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ZonalMapNumberFormat	Zonal Map Number Format. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ZonalMapLocation	Zonal Map Location. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ZonalMapFormat	Zonal Map Format. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AdaptiveMapFormat	Adaptive Map Format. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CodeNumberFormat	Code Number Format. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CodeLabel	Code Label. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfTables	Number of Tables. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfTable	Number of Tables. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CodeTableLocation	Code Table Location. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_BitsForCodeWord	Bits For Code Word. This tag is

	marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageDataLocation	Image Data Location. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PixelSpacingCalibrationType	Pixel Spacing Calibration Type.
DCM_TAG_PixelSpacingCalibrationDescription	Pixel Spacing Calibration Description.
DCM_TAG_PixelIntensityRelationship	Pixel Intensity Relationship.
DCM_TAG_PixelIntensityRelationshipSign	Pixel Intensity Relationship Sign.
DCM_TAG_WindowCenter	Window Center.
DCM_TAG_WindowWidth	Window Width.
DCM_TAG_RescaleIntercept	Rescale Intercept.
DCM_TAG_RescaleSlope	Rescale Slope.
DCM_TAG_RescaleType	Rescale Type.
DCM_TAG_WindowCenterWidthExplanation	Window Center and Width Explanation.
DCM_TAG_WindowCenterAndWidthExplanation	Window Center and Width Explanation. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_VOILUTFunction	VOI LUT Function.
DCM_TAG_GrayScale	Gray Scale. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RecommendedViewingMode	Recommended Viewing Mode.
DCM_TAG_GrayLookupTableDescriptor	Gray Lookup Table Descriptor. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RedPaletteColorLookupTableDescriptor	Red Palette Color Lookup Table Descriptor.
DCM_TAG_GreenPaletteColorLookupTableDescriptor	Green Palette Color Lookup Table Descriptor.
DCM_TAG_BluePaletteColorLookupTableDescriptor	Blue Palette Color Lookup Table Descriptor.
DCM_TAG_AlphaPaletteColorLookupTableDescriptor	Alpha Palette Color Lookup Table Descriptor.
DCM_TAG_LargeRedPaletteColorLookupTableDescriptor	Large Red Palette Color Lookup Table Descriptor. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LargeGreenPaletteColorLookupTableDescriptor	Large Green Palette Color Lookup Table Descriptor. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LargeBluePaletteColorLookupTableDescriptor	Large Blue Palette Color Lookup Table Descriptor. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.

DCM_TAG_PaletteColorLookupTableUID	Palette Color Lookup Table UID.
DCM_TAG_LutUID	Palette Color Lookup Table UID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_GrayLookupTableData	Gray Lookup Table Data. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RedPaletteColorLookupTableData	Red Palette Color Lookup Table Data.
DCM_TAG_GreenPaletteColorLookupTableData	Green Palette Color Lookup Table Data.
DCM_TAG_BluePaletteColorLookupTableData	Blue Palette Color Lookup Table Data.
DCM_TAG_AlphaPaletteColorLookupTableData	Alpha Palette Color Lookup Table Data.
DCM_TAG_LargeRedPaletteColorLookupTableData	Large Red Palette Color Lookup Table Data. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LargeGreenPaletteColorLookupTableData	Large Green Palette Color Lookup Table Data(RET). See DICOM specification for alternatives.
DCM_TAG_LargeBluePaletteColorLookupTableData	Large Blue Palette Color Lookup Table Data. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LargePaletteColorLookupTableUID	Large Palette Color Lookup Table UID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SegmentedRedPaletteColorLookupTableData	Segmented Red Palette Color Lookup Table Data.
DCM_TAG_SegmentedRedLutData	Segmented Red Palette Color Lookup Table Data.This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SegmentedGreenPaletteColorLookupTableData	Segmented Green Palette Color Lookup Table Data.
DCM_TAG_SegmentedGreenLutData	Segmented Green Palette Color Lookup Table Data. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SegmentedBluePaletteColorLookupTableData	Segmented Blue Palette Color Lookup Table Data.
DCM_TAG_SegmentedBlueLutData	Segmented Blue Palette Color Lookup Table Data. This tag name has been deprecated and

	will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BreastImplantPresent	Breast Implant Present.
DCM_TAG_ImplantPresent	Breast Implant Present. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PartialView	Partial View.
DCM_TAG_PartialViewDescription	Partial View Description.
DCM_TAG_PartialViewCodeSequence	Partial View Code Sequence.
DCM_TAG_SpatialLocationsPreserved	Spatial Locations Preserved.
DCM_TAG_DataFrameAssignmentSequence	Data Frame Assignment Sequence.
DCM_TAG_DataPathAssignment	Data Path Assignment.
DCM_TAG_BitsMappedToColorLookupTable	Bits Mapped to Color Lookup Table.
DCM_TAG_BlendingLUT1Sequence	Blending LUT 1 Sequence.
DCM_TAG_BlendingLUT1TransferFunction	Blending LUT 1 Transfer Function.
DCM_TAG_BlendingWeightConstant	Blending Weight Constant.
DCM_TAG_BlendingLookupTableDescriptor	Blending Lookup Table Descriptor.
DCM_TAG_BlendingLookupTableData	Blending Lookup Table Data.
DCM_TAG_EnhancedPaletteColorLookupTableSequence	Enhanced Palette Color Lookup Table Sequence.
DCM_TAG_BlendingLUT2Sequence	Blending LUT 2 Sequence.
DCM_TAG_BlendingLUT2TransferFunction	Blending LUT 2 Transfer Function.
DCM_TAG_DataPathID	Data Path ID.
DCM_TAG_RGBLUTTransferFunction	RGB LUT Transfer Function.
DCM_TAG_AlphaLUTTransferFunction	Alpha LUT Transfer Function.
DCM_TAG_ICCProfile	ICC Profile.
DCM_TAG_LossyImageCompression	Lossy Image Compression.
DCM_TAG_LossyImageCompressionRatio	Lossy Image Compression Ratio.
DCM_TAG_LossyImageCompressionMethod	Lossy Image Compression Method.
DCM_TAG_ModalityLUTSequence	Modality LUT Sequence.
DCM_TAG_LUTDescriptor	LUT Descriptor.
DCM_TAG_LUTExplanation	LUT Explanation.
DCM_TAG_ModalityLUTType	Modality LUT Type.
DCM_TAG_LUTData	LUT (Lookup Table) Data.
DCM_TAG_VOILUTSequence	VOI LUT Sequence.
DCM_TAG_SoftcopyVOILUTSequence	Softcopy VOI LUT Sequence.
DCM_TAG_ImagePresentationComments	Image Presentation Comments. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_BiPlaneAcquisitionSequence	Bi-Plane Acquisition Sequence. This tag is marked as retired in DICOM specification. See DICOM

	specification for alternatives.
DCM_TAG_RepresentativeFrameNumber	Representative Frame Number.
DCM_TAG_FrameNumbersOfInterest	Frame Numbers of Interest (FOI).
DCM_TAG_FrameOfInterestDescription	Frame of Interest Description.
DCM_TAG_FramesOfInterestDescription	Frame of Interest Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FrameofInterestType	Frame of Interest Type.
DCM_TAG_MaskPointers	Mask Pointer(s). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RWavePointer	R Wave Pointer.
DCM_TAG_RWavePoints	R Wave Pointer. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MaskSubtractionSequence	Mask Subtraction Sequence.
DCM_TAG_MaskOperation	Mask Operation.
DCM_TAG_ApplicableFrameRange	Applicable Frame Range.
DCM_TAG_MaskFrameNumbers	Mask Frame Numbers.
DCM_TAG_ContrastFrameAveraging	Contrast Frame Averaging.
DCM_TAG_MaskSubpixelShift	Mask Sub-pixel Shift.
DCM_TAG_TidOffset	TID Offset.
DCM_TAG_MaskOperationExplanation	Mask Operation Explanation.
DCM_TAG_PixelDataProviderURL	Pixel Data Provider URL.
DCM_TAG_DataPointRows	Data Point Rows.
DCM_TAG_DataPointColumns	Data Point Columns.
DCM_TAG_SignalDomainColumns	Signal Domain Columns.
DCM_TAG_LargestMonochromePixelValue	Largest Monochrome Pixel Value. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DataRepresentation	Data Representation.
DCM_TAG_PixelMeasuresSequence	Pixel Measures Sequence.
DCM_TAG_FrameVoiLutSequence	Frame VOI LUT Sequence.
DCM_TAG_PixelValueTransformationSequence	Pixel Value Transformation Sequence.
DCM_TAG_SignalDomainRows	Signal Domain Rows.
DCM_TAG_DisplayFilterPercentage	Display Filter Percentage.
DCM_TAG_FramePixelShiftSequence	Frame Pixel Shift Sequence.
DCM_TAG_SubtractionItemID	Subtraction Item ID.
DCM_TAG_PixelIntensityRelationshipLUTSequence	Pixel Intensity Relationship LUT Sequence.
DCM_TAG_FramePixelDataPropertiesSequence	Frame Pixel Data Properties Sequence.
DCM_TAG_GeometricalProperties	Geometrical Properties.

DCM_TAG_GeometricMaximumDistortion	Geometric Maximum Distortion.
DCM_TAG_ImageProcessingApplied	Image Processing Applied.
DCM_TAG_MaskSelectionMode	Mask Selection Mode.
DCM_TAG_LUTFunction	LUT Function.
DCM_TAG_MaskVisibilityPercentage	Mask Visibility Percentage.
DCM_TAG_PixelShiftSequence	Pixel Shift Sequence.
DCM_TAG_RegionPixelShiftSequence	Region Pixel Shift Sequence.
DCM_TAG_VerticesOfTheRegion	Vertices of the Region.
DCM_TAG_MultiFramePresentationSequence	Multi-frame Presentation Sequence.
DCM_TAG_PixelShiftFrameRange	Pixel Shift Frame Range.
DCM_TAG_LUTFrameRange	LUT Frame Range.
DCM_TAG_ImageToEquipmentMappingMatrix	Image to Equipment Mapping Matrix.
DCM_TAG_EquipmentCoordinateSystemIdentification	Equipment Coordinate System Identification.
DCM_TAG_Group0032Length	Group 0032 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyStatusID	Study Status ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyPriorityID	Study Priority ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyIDIssuer	Study ID Issuer. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyVerifiedDate	Study Verified Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyVerifiedTime	Study Verified Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyReadDate	Study Read Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyReadTime	Study Read Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ScheduledStudyStartDate	Scheduled Study Start Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ScheduledStudyStartTime	Scheduled Study Start Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ScheduledStudyStopDate	Scheduled Study Stop Date. This tag is marked as retired in DICOM specification. See DICOM

	specification for alternatives.
DCM_TAG_ScheduledStudyStopTime	Scheduled Study Stop Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ScheduledStudyLocation	Scheduled Study Location. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ScheduledStudyLocationAeTitles	Scheduled Study Location AE Title. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReasonForStudy	Reason for Study. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RequestingPhysicianIdentificationSequence	Requesting Physician Identification Sequence.
DCM_TAG_RequestingPhysicianIDSequence	Requesting Physician Identification Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RequestingPhysician	Requesting Physician.
DCM_TAG_RequestingService	Requesting Service.
DCM_TAG_RequestingServiceCodeSequence	Requesting Service Code Sequence.
DCM_TAG_StudyArrivalDate	Study Arrival Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyArrivalTime	Study Arrival Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyCompletionDate	Study Completion Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyCompletionTime	Study Completion Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StudyComponentStatusID	Study Component Status ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RequestedProcedureDescription	Requested Procedure Description.
DCM_TAG_RequestedProcedureCodeSequence	Requested Procedure Code Sequence.
DCM_TAG_RequestedContrastAgent	Requested Contrast Agent.
DCM_TAG_StudyComments	Study Comments. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.

DCM_TAG_Group0038Length	Group 0038 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedPatientAliasSequence	Referenced Patient Alias Sequence.
DCM_TAG_VisitStatusID	Visit Status ID.
DCM_TAG_AdmissionID	Admission ID.
DCM_TAG_IssuerOfAdmissionID	Issuer of Admission ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_IssuerOfAdmissionIDSequence	Issuer of Admission ID Sequence.
DCM_TAG_RouteOfAdmissions	Route of Admissions.
DCM_TAG_ScheduledAdmissionDate	Scheduled Admission Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ScheduledAdmissionTime	Scheduled Admission Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ScheduledDischargeDate	Scheduled Discharge Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ScheduledDischargeTime	Scheduled Discharge Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ScheduledPatientInstitutionResidence	Scheduled Patient Institution Residence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AdmittingDate	Admitting Date.
DCM_TAG_AdmittingTime	Admitting Time.
DCM_TAG_DischargeDate	Discharge Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DischargeTime	Discharge Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DischargeDiagnosisDescription	Discharge Diagnosis Description. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DischargeDiagnosisCodeSequence	Discharge Diagnosis Code Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SpecialNeeds	Special Needs.
DCM_TAG_ServiceEpisodeID	Service Episode ID.
DCM_TAG_IssuerOfServiceEpisodeID	Issuer of Service Episode ID. This tag is marked as retired in DICOM specification. See DICOM

	specification for alternatives.
DCM_TAG_ServiceEpisodeDescription	Service Episode Description.
DCM_TAG_IssuerOfServiceEpisodeIDSequence	Issuer of Service Episode ID Sequence.
DCM_TAG_PertinentDocumentsSequence	Pertinent Documents Sequence.
DCM_TAG_CurrentPatientLocation	Current Patient Location.
DCM_TAG_PatientInstitutionResidence	Patient's Institution Residence.
DCM_TAG_PatientsInstitutionResidence	Patient's Institution Residence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientState	Patient State.
DCM_TAG_PatientClinicalTrialParticipationSequence	Patient Clinical Trial Participation Sequence.
DCM_TAG_VisitComments	Visit Comments.
DCM_TAG_Group003ALength	Group 003A Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_WaveformOriginality	Waveform Originality.
DCM_TAG_NumberOfWaveformChannels	Number of Waveform Channels.
DCM_TAG_NumberOfWaveformSamples	Number of Waveform Samples.
DCM_TAG_SamplingFrequency	Sampling Frequency.
DCM_TAG_MultiplexGroupLabel	Multiplex Group Label.
DCM_TAG_GroupLabel	Multiplex Group Label. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_WaveformSampleValueRepresentation	Waveform Sample Value Representation.
DCM_TAG_ChannelDefinitionSequence	Channel Definition Sequence.
DCM_TAG_WaveformChannelNumber	Waveform Channel Number.
DCM_TAG_ChannelLabel	Channel Label.
DCM_TAG_ChannelStatus	Channel Status.
DCM_TAG_ChannelSourceSequence	Channel Source Sequence.
DCM_TAG_WaveformSource	Channel Source Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ChannelSourceModifiersSequence	Channel Source Modifiers Sequence.
DCM_TAG_WaveformSourceModifiers	Channel Source Modifiers Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SourceWaveformSequence	Source Waveform Sequence.

DCM_TAG_DifferentialWaveformSource	Source Waveform Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DifferentialWaveformSourceModifiers	Differential Waveform Source Modifiers. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ChannelDerivationDescription	Channel Derivation Description.
DCM_TAG_ChannelSensitivity	Channel Sensitivity.
DCM_TAG_ChannelSensitivityUnitsSequence	Channel Sensitivity Units Sequence.
DCM_TAG_ChannelSensitivityUnits	Channel Sensitivity Units Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ChannelSensitivityCorrectionFactor	Channel Sensitivity Correction Factor.
DCM_TAG_ChannelBaseline	Channel Baseline.
DCM_TAG_ChannelTimeSkew	Channel Time Skew.
DCM_TAG_ChannelSampleSkew	Channel Sample Skew.
DCM_TAG_ChannelOffset	Channel Offset.
DCM_TAG_WaveformBitsStored	Waveform Bits Stored.
DCM_TAG_FilterLowFrequency	Filter Low Frequency.
DCM_TAG_FilterHighFrequency	Filter High Frequency.
DCM_TAG_NotchFilterFrequency	Notch Filter Frequency.
DCM_TAG_NotchFilterBandwidth	Notch Filter Bandwidth.
DCM_TAG_WaveformDataDisplayScale	Waveform Data Display Scale.
DCM_TAG_WaveformDisplayBackgroundCIELabValue	Waveform Display Background CIELab Value.
DCM_TAG_WaveformPresentationGroupSequence	Waveform Presentation Group Sequence.
DCM_TAG_PresentationGroupNumber	Presentation Group Number.
DCM_TAG_ChannelDisplaySequence	Channel Display Sequence.
DCM_TAG_ChannelRecommendedDisplayCIELabValue	Channel Recommended Display CIELab Value.
DCM_TAG_ChannelPosition	Channel Position.
DCM_TAG_DisplayShadingFlag	Display Shading Flag.
DCM_TAG_FractionalChannelDisplayScale	Fractional Channel Display Scale.
DCM_TAG_AbsoluteChannelDisplayScale	Absolute Channel Display Scale.
DCM_TAG_MultiplexedAudioChannelsDescriptionCodeSequence	Multiplexed Audio Channels Description Code Sequence.
DCM_TAG_ChannelIdentificationCode	Channel Identification Code.
DCM_TAG_ChannelMode	Channel Mode.
DCM_TAG_Group0040Length	Group 0040 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.

DCM_TAG_ScheduledStationAeTitle	Scheduled Station AE Title.
DCM_TAG_ScheduledProcedureStepStartDate	Scheduled Procedure Step Start Date.
DCM_TAG_ScheduledProcedureStepStartTime	Scheduled Procedure Step Start Time.
DCM_TAG_ScheduledProcedureStepEndDate	Scheduled Procedure Step End Date.
DCM_TAG_ScheduledProcedureStepEndTime	Scheduled Procedure Step End Time.
DCM_TAG_ScheduledPerformingPhysiciansName	Scheduled Performing Physician's Name.
DCM_TAG_ScheduledPerformingPhysicianName	Scheduled Performing Physician's Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ScheduledProcedureStepDescription	Scheduled Procedure Step Description.
DCM_TAG_ScheduledProtocolCodeSequence	Scheduled Protocol Code Sequence.
DCM_TAG_ScheduledActionItemCodeSequence	Scheduled Protocol Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ScheduledProcedureStepID	Scheduled Procedure Step ID.
DCM_TAG_StageCodeSequence	Stage Code Sequence.
DCM_TAG_ScheduledPerformingPhysicianIdentificationSequence	Scheduled Performing Physician Identification Sequence.
DCM_TAG_ScheduledPerformingPhysicianIDSequence	Scheduled Performing Physician Identification Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ScheduledStationName	Scheduled Station Name.
DCM_TAG_ScheduledProcedureStepLocation	Scheduled Procedure Step Location.
DCM_TAG_PreMedication	Pre Medication.
DCM_TAG_ScheduledProcedureStepStatus	Scheduled Procedure Step Status.
DCM_TAG_OrderPlacerIdentifierSequence	Order Placer Identifier Sequence.
DCM_TAG_OrderFillerIdentifierSequence	Order Filler Identifier Sequence.
DCM_TAG_LocalNamespaceEntityID	Local Namespace Entity ID.
DCM_TAG_UniversalEntityID	Universal Entity ID.
DCM_TAG_UniversalEntityIDType	Universal Entity ID Type.
DCM_TAG_IdentifierTypeCode	Identifier Type Code.
DCM_TAG_AssigningFacilitySequence	Assigning Facility Sequence.
DCM_TAG_AssigningJurisdictionCodeSequence	Assigning Jurisdiction Code Sequence.
DCM_TAG_AssigningAgencyOrDepartmentCodeSequence	Assigning Agency or Department Code Sequence.

DCM_TAG_ScheduledProcedureStepSequence	Scheduled Procedure Step Sequence.
DCM_TAG_ReferencedNonImageCompositeSOPInstanceSequence	Referenced Non-Image Composite SOP Instance Sequence.
DCM_TAG_RefStandaloneSOPInstSequence	Referenced Non-Image Composite SOP Instance Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PerformedStationAeTitle	Performed Station AE Title.
DCM_TAG_PerformedStationName	Performed Station Name.
DCM_TAG_PerformedLocation	Performed Location.
DCM_TAG_PerformedProcedureStepStartDate	Performed Procedure Step Start Date.
DCM_TAG_PerformedProcedureStepStartTime	Performed Procedure Step Start Time.
DCM_TAG_PerformedProcedureStepEndDate	Performed Procedure Step End Date.
DCM_TAG_PerformedProcedureStepEndTime	Performed Procedure Step End Time.
DCM_TAG_PerformedProcedureStepStatus	Performed Procedure Step Status.
DCM_TAG_PerformedProcedureStepID	Performed Procedure Step ID.
DCM_TAG_PerformedProcedureStepDescription	Performed Procedure Step Description.
DCM_TAG_PerformedProcedureStepDesc	Performed Procedure Step Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PerformedProcedureTypeDescription	Performed Procedure Type Description.
DCM_TAG_PerformedProcedureTypeDesc	Performed Procedure Type Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PerformedProtocolCodeSequence	Performed Protocol Code Sequence.
DCM_TAG_PerformedActionItemSequence	Performed Protocol Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PerformedProtocolType	Performed Protocol Type.
DCM_TAG_ScheduledStepAttributesSequence	Scheduled Step Attributes Sequence.
DCM_TAG_RequestAttributesSequence	Request Attributes Sequence.
DCM_TAG_CommentsOnThePerformedProcedureStep	Comments on the Performed Procedure Step.

DCM_TAG_PerformedProcedureStepComment	Comments on the Performed Procedure Step. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PerformedProcedureStepDiscontinuationReasonCodeSequence	Performed Procedure Step Discontinuation Reason Code Sequence.
DCM_TAG_PerfProcStepDiscontReasonCodeSequence	Performed Procedure Step Discontinuation Reason Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_QuantitySequence	Quantity Sequence.
DCM_TAG_Quantity	Quantity value.
DCM_TAG_MeasuringUnitsSequence	Measuring Units Sequence.
DCM_TAG_MeasuringUnitSequence	Measuring Units Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BillingItemSequence	Billing Item Sequence.
DCM_TAG_TotalTimeofFluoroscopy	Total Time of Fluoroscopy.
DCM_TAG_TotalNumberOfExposures	Total Number of Exposures.
DCM_TAG_TotalNumberofExposure	Total Number of Exposures. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_EntranceDose	Entrance Dose.
DCM_TAG_ExposedArea	Exposed Area.
DCM_TAG_DistanceSourceToEntrance	Distance Source to Entrance.
DCM_TAG_SourceToEntranceDistance	Distance Source to Entrance. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DistanceSourceToSupport	Distance Source to Support. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ExposureDoseSequence	Exposure Dose Sequence.
DCM_TAG_CommentsOnRadiationDose	Comments on Radiation Dose.
DCM_TAG_RadiationDoseComment	Comments on Radiation Dose. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.

DCM_TAG_XrayOutput	X-Ray Output.
DCM_TAG_HalfValueLayer	Half Value Layer.
DCM_TAG_OrganDose	Organ Dose.
DCM_TAG_OrganExposed	Organ Exposed.
DCM_TAG_BillingProcedureStepSequence	Billing Procedure Step Sequence.
DCM_TAG_FilmConsumptionSequence	Film Consumption Sequence.
DCM_TAG_BillingSuppliesAndDevicesSequence	Billing Supplies and Devices Sequence.
DCM_TAG_BillingSuppliesAndDeviceSequence	Billing Supplies and Devices Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BillingSupplyDeviceSequence	Billing Supplies and Devices Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedProcedureStepSequence	Referenced Procedure Step Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PerformedSeriesSequence	Performed Series Sequence.
DCM_TAG_CommentsOnTheScheduledProcedureStep	Comments on the Scheduled Procedure Step.
DCM_TAG_ScheduledProcedureComment	Comments on the Scheduled Procedure Step. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ProtocolContextSequence	Protocol Context Sequence.
DCM_TAG_ContentItemModifierSequence	Content Item Modifier Sequence.
DCM_TAG_ScheduledSpecimenSequence	Scheduled Specimen Sequence.
DCM_TAG_SpecimenAccessionNumber	Specimen Accession Number. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ContainerIdentifier	Container Identifier.
DCM_TAG_IssuerOfTheContainerIdentifierSequence	Issuer of the Container Identifier Sequence.
DCM_TAG_AlternateContainerIdentifierSequence	Alternate Container Identifier Sequence.
DCM_TAG_ContainerTypeCodeSequence	Container Type Code Sequence.
DCM_TAG_ContainerDescription	Container Description.
DCM_TAG_ContainerComponentSequence	Container Component Sequence.
DCM_TAG_SpecimenSequence	Specimen Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SpecimenIdentifier	Specimen Identifier.
DCM_TAG_SpecimenDescriptionSequenceTrial	Specimen Description Sequence

	- Trial. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SpecimenDescriptionTrial	Specimen Description - Trial. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SpecimenDescription	Specimen Description - Trial. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SpecimenUID	Specimen UID.
DCM_TAG_AcquisitionContextSequence	Acquisition Context Sequence.
DCM_TAG_AcquisitionContextDescription	Acquisition Context Description.
DCM_TAG_AcquisitionContextDesc	Acquisition Context Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SpecimenDescriptionSequence	Specimen Description Sequence.
DCM_TAG_IssuerOfTheSpecimenIdentifierSequence	Issuer of the Specimen Identifier Sequence.
DCM_TAG_SpecimenTypeCodeSequence	Specimen Type Code Sequence.
DCM_TAG_SpecimenShortDescription	Specimen Short Description.
DCM_TAG_SpecimenDetailedDescription	Specimen Detailed Description.
DCM_TAG_SpecimenPreparationSequence	Specimen Preparation Sequence.
DCM_TAG_SpecimenPreparationStepContentItemSequence	Specimen Preparation Step Content Item Sequence.
DCM_TAG_SpecimenLocalizationContentItemSequence	Specimen Localization Content Item Sequence.
DCM_TAG_SlideIdentifier	Slide Identifier. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageCenterPointCoordinatesSequence	Image Center Point Coordinates Sequence.
DCM_TAG_ImageCenterPointCoordSequence	Image Center Point Coordinates Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_XOffsetInSlideCoordinateSystem	X offset in Slide Coordinate System.
DCM_TAG_XOffset	X offset in Slide Coordinate System. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_YOffsetInSlideCoordinateSystem	Y offset in Slide Coordinate System.
DCM_TAG_YOffset	Y offset in Slide Coordinate System. This tag name has been deprecated and will be removed

	from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ZOffsetInSlideCoordinateSystem	Z offset in Slide Coordinate System.
DCM_TAG_ZOffset	Z offset in Slide Coordinate System. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PixelSpacingSequence	Pixel Spacing Sequence.
DCM_TAG_CoordinateSystemAxisCodeSequence	Coordinate System Axis Code Sequence.
DCM_TAG_CoordSystemAxisCodeSequence	Coordinate System Axis Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MeasurementUnitsCodeSequence	Measurement Units Code Sequence.
DCM_TAG_MeasurementUnitCodeSequence	Measurement Units Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_VitalStainCodeSequenceTrial	Vital Stain Code Sequence - Trial. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_VitalStainCodeSequence	Vital Stain Code Sequence - Trial. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RequestedProcedureID	Requested Procedure ID.
DCM_TAG_ReasonForTheRequestedProcedure	Reason for the Requested Procedure.
DCM_TAG_RequestedProcedureReason	Reason for the Requested Procedure. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RequestedProcedurePriority	Requested Procedure Priority.
DCM_TAG_PatientTransportArrangements	Patient Transport Arrangements.
DCM_TAG_PatientTransport	Patient Transport Arrangements. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RequestedProcedureLocation	Requested Procedure Location.

DCM_TAG_PlacerOrderNumberProcedure	Placer Order Number / Procedure. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_FillerOrderNumberProcedure	Filler Order Number / Procedure. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ConfidentialityCode	Confidentiality Code.
DCM_TAG_ReportingPriority	Reporting Priority.
DCM_TAG_ReasonforRequestedProcedureCodeSequence	Reason for Requested Procedure Code Sequence.
DCM_TAG_NamesOfIntendedRecipientsOfResults	Names of Intended Recipients of Results.
DCM_TAG_IntendedRecipients	Names of Intended Recipients of Results. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_IntendedRecipientsOfResultsIdentificationSequence	Intended Recipients of Results Identification Sequence.
DCM_TAG_IntendedRecipientOfResultIDSequence	Intended Recipients of Results Identification Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReasonForPerformedProcedureCodeSequence	Reason For Performed Procedure Code Sequence.
DCM_TAG_PersonIdentificationCodeSequence	Person Identification Code Sequence.
DCM_TAG_PersonIDCodeSequence	Person Identification Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PersonsAddress	Person's Address.
DCM_TAG_PersonAddress	Person's Address. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PersonTelephoneNumbers	Person's Telephone Numbers.
DCM_TAG_PersonsTelephoneNumbers	Person's Telephone Numbers. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PersonTelephoneNumber	Person's Telephone Numbers. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with

	the same value defined in the previous line.
DCM_TAG_RequestedProcedureComments	Requested Procedure Comments.
DCM_TAG_RequestedProcedureComment	Requested Procedure Comments. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReasonForTheImagingServiceRequest	Reason for the Imaging Service Request. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_IssueDateOfImagingServiceRequest	Issue Date of Imaging Service Request.
DCM_TAG_ImagingServiceRequestDate	Issue Date of Imaging Service Request. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_IssueTimeOfImagingServiceRequest	Issue Time of Imaging Service Request.
DCM_TAG_ImagingServiceRequestTime	Issue Time of Imaging Service Request. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PlacerOrderNumberImagingServiceRequestRetired	Placer Order Number / Imaging Service Request (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImagingServiceRequestPlacerOrderNum	Placer Order Number / Imaging Service Request (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_FillerOrderNumberImagingServiceRequestRetired	Filler Order Number / Imaging Service Request (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImagingServiceRequestFillerOrderNum	Filler Order Number / Imaging Service Request (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OrderEnteredBy	Order Entered By.
DCM_TAG_OrderEnterer	Order Entered By. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OrderEnterersLocation	Order Enterer's Location.
DCM_TAG_OrderEntererLocation	Order Enterer's Location. This

	tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OrderCallbackPhoneNumber	Order Callback Phone Number.
DCM_TAG_PlacerOrderNumberImagingServiceRequest	Placer Order Number / Imaging Service Request.
DCM_TAG_FillerOrderNumberImagingServiceRequest	Filler Order Number / Imaging Service Request.
DCM_TAG_ImagingServiceRequestComments	Imaging Service Request Comments.
DCM_TAG_ImagingServiceRequestComment	Imaging Service Request Comments. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ConfidentialityConstraintOnPatientDataDescription	Confidentiality Constraint on Patient Data Description.
DCM_TAG_ConfidentialityConstraint	Confidentiality Constraint on Patient Data Description.
DCM_TAG_GeneralPurposeScheduledProcedureStepStatus	General Purpose Scheduled Procedure Step Status.
DCM_TAG_GeneralScheduledProcedureStepStatus	General Purpose Scheduled Procedure Step Status. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_GeneralPurposePerformedProcedureStepStatus	General Purpose Performed Procedure Step Status.
DCM_TAG_GeneralPerformedProcedureStepStatus	General Purpose Performed Procedure Step Status. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_GeneralPurposeScheduledProcedureStepPriority	General Purpose Scheduled Procedure Step Priority.
DCM_TAG_GeneralScheduledProcedureStepPriority	General Purpose Scheduled Procedure Step Priority. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ScheduledProcessingApplicationsCodeSequence	Scheduled Processing Applications Code Sequence.
DCM_TAG_ScheduledProcessingAppCodeSequence	Scheduled Processing Applications Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ScheduledProcedureStepStartDateTime	Scheduled Procedure Step Start

	Date Time.
DCM_TAG_ScheduledProcedureStepStartDateAndTime	Scheduled Procedure Step Start Date and Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ScheduledProcedureStepStartDT	Scheduled Procedure Step Start Date and Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MultipleCopiesFlag	Multiple Copies Flag.
DCM_TAG_PerformedProcessingApplicationsCodeSequence	Performed Processing Applications Code Sequence.
DCM_TAG_PerformedProcessingAppCodeSequence	Performed Processing Applications Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_HumanPerformerCodeSequence	Human Performer Code Sequence.
DCM_TAG_ScheduledProcedureStepModificationDateTime	Scheduled Procedure Step Modification Date Time.
DCM_TAG_ScheduledProcedureStepModificationDateandTime	Scheduled Procedure Step Modification Date and Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ExpectedCompletionDateTime	Expected Completion Date Time.
DCM_TAG_ExpectedCompletionDateAndTime	Expected Completion Date and Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ExpectedCompletionDT	Expected Completion Date and Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ResultingGeneralPurposePerformedProcedureStepsSequence	Resulting General Purpose Performed Procedure Steps Sequence.
DCM_TAG_ResultingGenPerformedProcStepSequence	Resulting General Purpose Performed Procedure Steps Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined

	in the previous line.
DCM_TAG_ReferencedGeneralPurposeScheduledProcedureStepSequence	Referenced General Purpose Scheduled Procedure Step Sequence.
DCM_TAG_RefGenScheduledProcStepSequence	Referenced General Purpose Scheduled Procedure Step Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ScheduledWorkitemCodeSequence	Scheduled Workitem Code Sequence.
DCM_TAG_PerformedWorkitemCodeSequence	Performed Workitem Code Sequence.
DCM_TAG_InputAvailabilityFlag	Input Availability Flag.
DCM_TAG_InputInformationSequence	Input Information Sequence.
DCM_TAG_RelevantInformationSequence	Relevant Information Sequence.
DCM_TAG_ReferencedGeneralPurposeScheduledProcedureStepTransactionUID	Referenced General Purpose Scheduled Procedure Step Transaction UID.
DCM_TAG_ReferencedGenScheduledProcStepUID	Referenced General Purpose Scheduled Procedure Step Transaction UID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ScheduledStationNameCodeSequence	Scheduled Station Name Code Sequence.
DCM_TAG_ScheduledStationClassCodeSequence	Scheduled Station Class Code Sequence.
DCM_TAG_ScheduledStationGeographicLocationCodeSequence	Scheduled Station Geographic Location Code Sequence.
DCM_TAG_ScheduledStationGeoLocationCode	Scheduled Station Geographic Location Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PerformedStationNameCodeSequence	Performed Station Name Code Sequence.
DCM_TAG_PerformedStationClassCodeSequence	Performed Station Class Code Sequence.
DCM_TAG_PerformedStationGeographicLocationCodeSequence	Performed Station Geographic Location Code Sequence.
DCM_TAG_PerformedStationGeoLocationCode	Performed Station Geographic Location Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RequestedSubsequentWorkitemCodeSequence	Requested Subsequent Workitem Code Sequence.

DCM_TAG_RequestedSubsWorkitemCodeSequence	Requested Subsequent Workitem Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_NonDICOMOutputCodeSequence	Non-DICOM Output Code Sequence.
DCM_TAG_OutputInformationSequence	Output Information Sequence.
DCM_TAG_ScheduledHumanPerformersSequence	Scheduled Human Performers Sequence.
DCM_TAG_ScheduledHumanPerformerSequence	Scheduled Human Performers Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ActualHumanPerformersSequence	Actual Human Performers Sequence.
DCM_TAG_ActualHumanPerformerSequence	Actual Human Performers Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_HumanPerformerOrganization	Human Performer's Organization.
DCM_TAG_HumanPerformersOrganization	Human Performer's Organization. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_HumanPerformerName	Human Performer's Name.
DCM_TAG_HumanPerformersName	Human Performer's Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RawDataHandling	Raw Data Handling.
DCM_TAG_EntranceDoseInmGy	Entrance Dose in mGy.
DCM_TAG_ReferencedImageRealWorldValueMappingSequence	Referenced Image Real World Value Mapping Sequence.
DCM_TAG_RealWorldValueMappingSequence	Real World Value Mapping Sequence.
DCM_TAG_RealWorldValMappingSequence	Real World Value Mapping Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PixelValueMappingCodeSequence	Pixel Value Mapping Code Sequence.
DCM_TAG_LutLabel	LUT Label.
DCM_TAG_RealWorldValueLastValueMapped	Real World Value Last Value

	Mapped.
DCM_TAG_RealWorldValLastValMapped	Real World Value Last Value Mapped. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RealWorldValueLUTData	Real World Value LUT Data.
DCM_TAG_RealWorldValLutData	Real World Value LUT Data. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RealWorldValueFirstValueMapped	Real World Value First Value Mapped.
DCM_TAG_RealWorldValFirstValMapped	Real World Value First Value Mapped. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RealWorldValueIntercept	Real World Value Intercept.
DCM_TAG_RealWorldValIntercept	Real World Value Intercept. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RealWorldValueSlope	Real World Value Slope.
DCM_TAG_RealWorldValSlope	Real World Value Slope. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RelationshipType	Relationship Type.
DCM_TAG_VerifyingOrganization	Verifying Organization.
DCM_TAG_VerificationDateTime	Verification Date Time.
DCM_TAG_ObservationDateTime	Observation Date Time.
DCM_TAG_ValueType	Value Type.
DCM_TAG_ConceptNameCodeSequence	Concept Name Code Sequence.
DCM_TAG_ContinuityOfContent	Continuity Of Content.
DCM_TAG_AlertCodeSequence	Alert Code Sequence.
DCM_TAG_VerifyingObserverSequence	Verifying Observer Sequence.
DCM_TAG_VerifyingObserverName	Verifying Observer Name.
DCM_TAG_AuthorObserverSequence	Author Observer Sequence.
DCM_TAG_ParticipantSequence	Participant Sequence.
DCM_TAG_CustodialOrganizationSequence	Custodial Organization Sequence.
DCM_TAG_ParticipationType	Participation Type.
DCM_TAG_ParticipationDatetime	Participation DateTime. This tag name has been deprecated and will be removed from the public

	API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ObserverType	Observer Type.
DCM_TAG_VerifyingObserverIdentificationCodeSequence	Verifying Observer Identification Code Sequence.
DCM_TAG_VerifyingObserverIDCodeSequence	Verifying Observer Identification Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_EquivalentCDADocumentSequence	Equivalent CDA Document Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedTypeofData	Referenced Type of Data.
DCM_TAG_ReferencedWaveformChannels	Referenced Waveform Channels.
DCM_TAG_DateTime	DateTime value.
DCM_TAG_Date	Date value.
DCM_TAG_Time	Time value.
DCM_TAG_PersonName	Person Name.
DCM_TAG_UID	UID (unique identifier).
DCM_TAG_TemporalRangeType	Temporal Range Type.
DCM_TAG_ReferencedSamplePositions	Referenced Sample Positions.
DCM_TAG_ReferencedSampleOffsets	Referenced Sample Positions. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedFrameNumbers	Referenced Frame Numbers.
DCM_TAG_ReferencedTimeOffsets	Referenced Time Offsets.
DCM_TAG_ReferencedDatetime	Referenced DateTime. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TextValue	Text Value.
DCM_TAG_ConceptCodeSequence	Concept Code Sequence.
DCM_TAG_PurposeOfReferenceCodeSequence	Purpose of Reference Code Sequence.
DCM_TAG_AnnotationGroupNumber	Annotation Group Number.
DCM_TAG_ModifierCodeSequence	Modifier Code Sequence.
DCM_TAG_ConceptCodeSequenceModifier	Modifier Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CoordinateGeometricType	Coordinate Geometric Type.
DCM_TAG_PixelCoordinateSet	Pixel Coordinate Set.
DCM_TAG_MeasuredValueSequence	Measured Value Sequence.

DCM_TAG_NumericValueQualifierCodeSequence	Numeric Value Qualifier Code Sequence.
DCM_TAG_NumericValue	Numeric Value.
DCM_TAG_AddressTrial	Address - Trial. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Address	Address - Trial. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TelephoneNumberTrial	Telephone Number - Trial. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TelephoneNumber	Telephone Number - Trial. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PredecessorDocumentsSequence	Predecessor Documents Sequence.
DCM_TAG_ReferencedRequestSequence	Referenced Request Sequence.
DCM_TAG_PerformedProcedureCodeSequence	Performed Procedure Code Sequence.
DCM_TAG_CurrentRequestedProcedureEvidenceSequence	Current Requested Procedure Evidence Sequence.
DCM_TAG_PertinentOtherEvidenceSequence	Pertinent Other Evidence Sequence.
DCM_TAG_HL7StructuredDocumentReferenceSequence	HL7 Structured Document Reference Sequence.
DCM_TAG_CompletionFlag	Completion Flag.
DCM_TAG_CompletionFlagDescription	Completion Flag Description.
DCM_TAG_VerificationFlag	Verification Flag.
DCM_TAG_ArchiveRequested	Archive Requested.
DCM_TAG_PreliminaryFlag	Preliminary Flag.
DCM_TAG_ContentTemplateSequence	Content Template Sequence.
DCM_TAG_IdenticalDocumentsSequence	Identical Documents Sequence.
DCM_TAG_ContentSequence	Content Sequence.
DCM_TAG_RelationshipTypeCodeSequence	Relationship Type Code Sequence.
DCM_TAG_WaveformAnnotationSequence	Waveform Annotation Sequence.
DCM_TAG_AnnotationSequence	Waveform Annotation Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TemplateIdentifier	Template Identifier.
DCM_TAG_TemplateVersion	Template Version. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TemplateLocalVersion	Template Local Version. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.

DCM_TAG_TemplateExtensionFlag	Template Extension Flag. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TemplateExtensionOrganizationUID	Template Extension Organization UID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TemplateExtensionCreatorUID	Template Extension Creator UID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedContentItemIdentifier	Referenced Content Item Identifier.
DCM_TAG_ReferencedContentItemID	Referenced Content Item Identifier. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_HL7InstanceIdentifier	HL7 Instance Identifier.
DCM_TAG_HL7DocumentEffectiveTime	HL7 Document Effective Time.
DCM_TAG_HL7DocumentTypeCodeSequence	HL7 Document Type Code Sequence.
DCM_TAG_RetrieveURI	Retrieve URI.
DCM_TAG_RetrieveLocationUID	Retrieve Location UID.
DCM_TAG_Group0042Length	Group 0042 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DocumentTitle	Document Title.
DCM_TAG_EncapsulatedDocument	Encapsulated Document.
DCM_TAG_MIMETypeofEncapsulatedDocument	MIME Type of Encapsulated Document.
DCM_TAG_SourceInstanceSequence	Source Instance Sequence.
DCM_TAG_ListOfMIMETypes	List of MIME Types.
DCM_TAG_Group0044Length	Group 0044 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ProductPackageIdentifier	Product Package Identifier.
DCM_TAG_SubstanceAdministrationApproval	Substance Administration Approval.
DCM_TAG_ApprovalStatusFurtherDescription	Approval Status Further Description.
DCM_TAG_ApprovalStatusDateTime	Approval Status DateTime.
DCM_TAG_ProductTypeCodeSequence	Product Type Code Sequence.
DCM_TAG_ProductName	Product Name.
DCM_TAG_ProductDescription	Product Description.
DCM_TAG_ProductLotIdentifier	Product Lot Identifier.
DCM_TAG_ProductExpirationDateTime	Product Expiration DateTime.
DCM_TAG_SubstanceAdministrationDateTime	Substance Administration DateTime.
DCM_TAG_SubstanceAdministrationNotes	Substance Administration Notes.

DCM_TAG_SubstanceAdministrationDeviceID	Substance Administration Device ID.
DCM_TAG_ProductParameterSequence	Product Parameter Sequence.
DCM_TAG_SubstanceAdministrationParameterSequence	Substance Administration Parameter Sequence.
DCM_TAG_Group0046Length	Group 0046 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LensDescription	Lens Description.
DCM_TAG_RightLensSequence	Right Lens Sequence.
DCM_TAG_LeftLensSequence	Left Lens Sequence.
DCM_TAG_UnspecifiedLateralityLensSequence	Unspecified Laterality Lens Sequence.
DCM_TAG_CylinderSequence	Cylinder Sequence.
DCM_TAG_PrismSequence	Prism Sequence.
DCM_TAG_HorizontalPrismPower	Horizontal Prism Power.
DCM_TAG_HorizontalPrismBase	Horizontal Prism Base.
DCM_TAG_VerticalPrismPower	Vertical Prism Power.
DCM_TAG_VerticalPrismBase	Vertical Prism Base.
DCM_TAG_LensSegmentType	Lens Segment Type.
DCM_TAG_OpticalTransmittance	Optical Transmittance.
DCM_TAG_ChannelWidth	Channel Width.
DCM_TAG_PupilSize	Pupil Size.
DCM_TAG_CornealSize	Corneal Size.
DCM_TAG_AutorefractionRightEyeSequence	Autorefraction Right Eye Sequence.
DCM_TAG_AutorefractionLeftEyeSequence	Autorefraction Left Eye Sequence.
DCM_TAG_DistancePupillaryDistance	Distance Pupillary Distance.
DCM_TAG_NearPupillaryDistance	Near Pupillary Distance.
DCM_TAG_IntermediatePupillaryDistance	Intermediate Pupillary Distance.
DCM_TAG_OtherPupillaryDistance	Other Pupillary Distance.
DCM_TAG_KeratometryRightEyeSequence	Keratometry Right Eye Sequence.
DCM_TAG_KeratometryLeftEyeSequence	Keratometry Left Eye Sequence.
DCM_TAG_SteepKeratometricAxisSequence	Steep Keratometric Axis Sequence.
DCM_TAG_RadiusOfCurvature	Radius of Curvature.
DCM_TAG_KeratometricPower	Keratometric Power.
DCM_TAG_KeratometricAxis	Keratometric Axis.
DCM_TAG_FlatKeratometricAxisSequence	Flat Keratometric Axis Sequence.
DCM_TAG_BackgroundColor	Background Color.
DCM_TAG_Optotype	Optotype tag.
DCM_TAG_OptotypePresentation	Optotype Presentation.
DCM_TAG_SubjectiveRefractionRightEyeSequence	Subjective Refraction Right Eye Sequence.
DCM_TAG_SubjectiveRefractionLeftEyeSequence	Subjective Refraction Left Eye Sequence.
DCM_TAG_AddNearSequence	Add Near Sequence.
DCM_TAG_AddIntermediateSequence	Add Intermediate Sequence.

DCM_TAG_AddOtherSequence	Add Other Sequence.
DCM_TAG_AddPower	Add Power.
DCM_TAG_ViewingDistance	Viewing Distance.
DCM_TAG_VisualAcuityTypeCodeSequence	Visual Acuity Type Code Sequence.
DCM_TAG_VisualAcuityRightEyeSequence	Visual Acuity Right Eye Sequence.
DCM_TAG_VisualAcuityLeftEyeSequence	Visual Acuity Left Eye Sequence.
DCM_TAG_VisualAcuityBothEyesOpenSequence	Visual Acuity Both Eyes Open Sequence.
DCM_TAG_ViewingDistanceType	Viewing Distance Type.
DCM_TAG_VisualAcuityModifiers	Visual Acuity Modifiers.
DCM_TAG_DecimalVisualAcuity	Decimal Visual Acuity.
DCM_TAG_OptotypeDetailedDefinition	Optotype Detailed Definition.
DCM_TAG_ReferencedRefractiveMeasurementsSequence	Referenced Refractive Measurements Sequence.
DCM_TAG_SpherePower	Sphere Power.
DCM_TAG_CylinderPower	Cylinder Power.
DCM_TAG_Group0050Length	Group 0050 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CalibrationImage	Calibration Image.
DCM_TAG_DeviceSequence	Device Sequence.
DCM_TAG_ContainerComponentTypeCodeSequence	Container Component Type Code Sequence.
DCM_TAG_ContainerComponentThickness	Container Component Thickness.
DCM_TAG_DeviceLength	Device Length.
DCM_TAG_ContainerComponentWidth	Container Component Width.
DCM_TAG_DeviceDiameter	Device Diameter.
DCM_TAG_DeviceDiameterUnits	Device Diameter Units.
DCM_TAG_DeviceDiameterUnit	Device Diameter Units. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DeviceVolume	Device Volume.
DCM_TAG_InterMarkerDistance	Inter-Marker Distance.
DCM_TAG_ContainerComponentMaterial	Container Component Material.
DCM_TAG_ContainerComponentID	Container Component ID.
DCM_TAG_ContainerComponentLength	Container Component Length.
DCM_TAG_ContainerComponentDiameter	Container Component Diameter.
DCM_TAG_ContainerComponentDescription	Container Component Description.
DCM_TAG_DeviceDescription	Device Description.
DCM_TAG_Group0054Length	Group 0054 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_EnergyWindowVector	Energy Window Vector.
DCM_TAG_NumberOfEnergyWindows	Number of Energy Windows.
DCM_TAG_EnergyWindowInformationSequence	Energy Window Information

	Sequence.
DCM_TAG_EnergyWindowInfoSequence	Energy Window Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_EnergyWindowRangeSequence	Energy Window Range Sequence.
DCM_TAG_EnergyWindowLowerLimit	Energy Window Lower Limit.
DCM_TAG_EnergyWindowUpperLimit	Energy Window Upper Limit.
DCM_TAG_RadiopharmaceuticalInformationSequence	Radiopharmaceutical Information Sequence.
DCM_TAG_RadiopharmaInfoSequence	Radiopharmaceutical Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ResidualSyringeCounts	Residual Syringe Counts.
DCM_TAG_EnergyWindowName	Energy Window Name.
DCM_TAG_DetectorVector	Detector Vector.
DCM_TAG_NumberOfDetectors	Number of Detectors.
DCM_TAG_DetectorInformationSequence	Detector Information Sequence.
DCM_TAG_DetectorInfoSequence	Detector Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PhaseVector	Phase Vector.
DCM_TAG_NumberOfPhases	Number of Phases.
DCM_TAG_PhaseInformationSequence	Phase Information Sequence.
DCM_TAG_PhaseInfoSequence	Phase Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_NumberOfFramesInPhase	Number of Frames in Phase.
DCM_TAG_PhaseDelay	Phase Delay.
DCM_TAG_PauseBetweenFrames	Pause Between Frames.
DCM_TAG_PhaseDescription	Phase Description.
DCM_TAG_RotationVector	Rotation Vector.
DCM_TAG_NumberOfRotations	Number of Rotations.
DCM_TAG_RotationInformationSequence	Rotation Information Sequence.
DCM_TAG_RotationInfoSequence	Rotation Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_NumberOfFramesInRotation	Number of Frames in Rotation.

DCM_TAG_RrIntervalVector	R-R Interval Vector.
DCM_TAG_NumberOfRrIntervals	Number of R-R Intervals.
DCM_TAG_GatedInformationSequence	Gated Information Sequence.
DCM_TAG_GatedInfoSequence	Gated Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DataInformationSequence	Data Information Sequence.
DCM_TAG_DataInfoSequence	Data Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TimeSlotVector	Time Slot Vector.
DCM_TAG_NumberOfTimeSlots	Number of Time Slots.
DCM_TAG_TimeSlotInformationSequence	Time Slot Information Sequence.
DCM_TAG_TimeSlotInfoSequence	Time Slot Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TimeSlotTime	Time Slot Time.
DCM_TAG_SliceVector	Slice Vector.
DCM_TAG_NumberOfSlices	Number of Slices.
DCM_TAG_AngularViewVector	Angular View Vector.
DCM_TAG_TimeSliceVector	Time Slice Vector.
DCM_TAG_NumberOfTimeSlices	Number of Time Slices.
DCM_TAG_StartAngle	Start Angle.
DCM_TAG_StartAngleVector	Start Angle. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TypeOfDetectorMotion	Type of Detector Motion.
DCM_TAG_TriggerVector	Trigger Vector.
DCM_TAG_NumberOfTriggersInPhase	Number of Triggers in Phase.
DCM_TAG_ViewCodeSequence	View Code Sequence.
DCM_TAG_ViewCodeSequnce	View Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ViewModifierCodeSequence	View Modifier Code Sequence.
DCM_TAG_ViewAngulationModifierCodeSequence	View Modifier Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the

	previous line.
DCM_TAG_RadionuclideCodeSequence	Radionuclide Code Sequence.
DCM_TAG_AdministrationRouteCodeSequence	Administration Route Code Sequence.
DCM_TAG_RadiopharmaceuticalCodeSequence	Radiopharmaceutical Code Sequence.
DCM_TAG_RadiopharmaCodeSequence	Radiopharmaceutical Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CalibrationDataSequence	Calibration Data Sequence.
DCM_TAG_EnergyWindowNumber	Energy Window Number.
DCM_TAG_ImageID	Image Identifier.
DCM_TAG_PatientOrientationCodeSequence	Patient Orientation Code Sequence.
DCM_TAG_PatientOrientCodeSequence	Patient Orientation Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientOrientationModifierCodeSequence	Patient Orientation Modifier Code Sequence.
DCM_TAG_PatientOrientModifierCodeSequence	Patient Orientation Modifier Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PatientGantryRelationshipCodeSequence	Patient Gantry Relationship Code Sequence.
DCM_TAG_PatientGantryRelationCodeSequence	Patient Gantry Relationship Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SliceProgressionDirection	Slice Progression Direction.
DCM_TAG_SeriesType	Series Type.
DCM_TAG_Units	The Units.
DCM_TAG_CountsSource	Counts Source.
DCM_TAG_ReprojectionMethod	Reprojection Method.
DCM_TAG_RandomsCorrectionMethod	Randoms Correction Method.
DCM_TAG_AttenuationCorrectionMethod	Attenuation Correction Method.
DCM_TAG_DecayCorrection	Decay Correction.
DCM_TAG_ReconstructionMethod	Reconstruction Method.
DCM_TAG_DetectorLinesOfResponseUsed	Detector Lines of Response Used.
DCM_TAG_ScatterCorrectionMethod	Scatter Correction Method.
DCM_TAG_AxialAcceptance	Axial Acceptance.
DCM_TAG_AxialMash	Axial Mash.
DCM_TAG_TransverseMash	Transverse Mash.

DCM_TAG_DetectorElementSize	Detector Element Size.
DCM_TAG_CoincidenceWindowWidth	Coincidence Window Width.
DCM_TAG_SecondaryCountsType	Secondary Counts Type.
DCM_TAG_FrameReferenceTime	Frame Reference Time.
DCM_TAG_PrimaryPromptsCountsAccumulated	Primary (Prompts) Counts Accumulated.
DCM_TAG_PrimaryCountsAccumulated	Primary (Prompts) Counts Accumulated. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SecondaryCountsAccumulated	Secondary Counts Accumulated.
DCM_TAG_SliceSensitivityFactor	Slice Sensitivity Factor.
DCM_TAG_DecayFactor	Decay Factor.
DCM_TAG_DoseCalibrationFactor	Dose Calibration Factor.
DCM_TAG_ScatterFractionFactor	Scatter Fraction Factor.
DCM_TAG_DeadTimeFactor	Dead Time Factor.
DCM_TAG_ImageIndex	Image Index.
DCM_TAG_CountsIncluded	Counts Included. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DeadTimeCorrectionFlag	Dead Time Correction Flag. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group0060Length	Group 0060 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_HistogramSequence	Histogram Sequence.
DCM_TAG_HistogramNumberOfBins	Histogram Number of Bins.
DCM_TAG_HistogramBinNumber	Histogram Number of Bins. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_HistogramFirstBinValue	Histogram First Bin Value.
DCM_TAG_HistogramLastBinValue	Histogram Last Bin Value.
DCM_TAG_HistogramBinWidth	Histogram Bin Width.
DCM_TAG_HistogramExplanation	Histogram Explanation.
DCM_TAG_HistogramData	Histogram Data.
DCM_TAG_Group0062Length	Group 0062 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SegmentationType	Segmentation Type.
DCM_TAG_SegmentSequence	Segment Sequence.
DCM_TAG_SegmentedPropertyCategoryCodeSequence	Segmented Property Category Code Sequence.
DCM_TAG_SegmentNumber	Segment Number.
DCM_TAG_SegmentLabel	Segment Label.

DCM_TAG_SegmentDescription	Segment Description.
DCM_TAG_SegmentAlgorithmType	Segment Algorithm Type.
DCM_TAG_SegmentAlgorithmName	Segment Algorithm Name.
DCM_TAG_SegmentIdentificationSequence	Segment Identification Sequence.
DCM_TAG_ReferencedSegmentNumber	Referenced Segment Number.
DCM_TAG_RecommendedDisplayGrayscaleValue	Recommended Display Grayscale Value.
DCM_TAG_RecommendedDisplayCIELabValue	Recommended Display CIELab Value.
DCM_TAG_MaximumFractionalValue	Maximum Fractional Value.
DCM_TAG_SegmentedPropertyTypeCodeSequence	Segmented Property Type Code Sequence.
DCM_TAG_SegmentationFractionalType	Segmentation Fractional Type.
DCM_TAG_Group0064Length	Group 0064 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DeformableRegistrationSequence	Deformable Registration Sequence.
DCM_TAG_SourceFrameOfReferenceUID	Source Frame of Reference UID.
DCM_TAG_DeformableRegistrationGridSequence	Deformable Registration Grid Sequence.
DCM_TAG_GridDimensions	Grid Dimensions.
DCM_TAG_GridResolution	Grid Resolution.
DCM_TAG_VectorGridData	Vector Grid Data.
DCM_TAG_PreDeformationMatrixRegistrationSequence	Pre Deformation Matrix Registration Sequence.
DCM_TAG_PostDeformationMatrixRegistrationSequence	Post Deformation Matrix Registration Sequence.
DCM_TAG_Group0066Length	Group 0066 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfSurfaces	Number of Surfaces.
DCM_TAG_SurfaceSequence	Surface Sequence.
DCM_TAG_SurfaceNumber	Surface Number.
DCM_TAG_SurfaceComments	Surface Comments.
DCM_TAG_SurfaceProcessing	Surface Processing.
DCM_TAG_SurfaceProcessingRatio	Surface Processing Ratio.
DCM_TAG_SurfaceProcessingDescription	Surface Processing Description.
DCM_TAG_RecommendedPresentationOpacity	Recommended Presentation Opacity.
DCM_TAG_RecommendedPresentationType	Recommended Presentation Type.
DCM_TAG_FiniteVolume	Finite Volume.
DCM_TAG_Manifold	Manifold tag.
DCM_TAG_SurfacePointsSequence	Surface Points Sequence.
DCM_TAG_SurfacePointsNormalsSequence	Surface Points Normals Sequence.
DCM_TAG_SurfaceMeshPrimitivesSequence	Surface Mesh Primitives Sequence.
DCM_TAG_NumberOfSurfacePoints	Number of Surface Points.

DCM_TAG_PointCoordinatesData	Point Coordinates Data.
DCM_TAG_PointPositionAccuracy	Point Position Accuracy.
DCM_TAG_MeanPointDistance	Mean Point Distance.
DCM_TAG_MaximumPointDistance	Maximum Point Distance.
DCM_TAG_PointsBoundingBoxCoordinates	Points Bounding Box Coordinates.
DCM_TAG_AxisOfRotation	Axis of Rotation.
DCM_TAG_CenterOfRotation	Center of Rotation.
DCM_TAG_NumberOfVectors	Number of Vectors.
DCM_TAG_VectorDimensionality	Vector Dimensionality.
DCM_TAG_VectorAccuracy	Vector Accuracy.
DCM_TAG_VectorCoordinateData	Vector Coordinate Data.
DCM_TAG_TrianglePointIndexList	Triangle Point Index List.
DCM_TAG_EdgePointIndexList	Edge Point Index List.
DCM_TAG_VertexPointIndexList	Vertex Point Index List.
DCM_TAG_TriangleStripSequence	Triangle Strip Sequence.
DCM_TAG_TriangleFanSequence	Triangle Fan Sequence.
DCM_TAG_LineSequence	Line Sequence.
DCM_TAG_PrimitivePointIndexList	Primitive Point Index List.
DCM_TAG_SurfaceCount	Surface Count.
DCM_TAG_ReferencedSurfaceSequence	Referenced Surface Sequence.
DCM_TAG_ReferencedSurfaceNumber	Referenced Surface Number.
DCM_TAG_SegmentSurfaceGenerationAlgorithmIdentificationSequence	Segment Surface Generation Algorithm Identification Sequence.
DCM_TAG_SegmentSurfaceSourceInstanceSequence	Segment Surface Source Instance Sequence.
DCM_TAG_AlgorithmFamilyCodeSequence	Algorithm Family Code Sequence.
DCM_TAG_AlgorithmNameCodeSequence	Algorithm Name Code Sequence.
DCM_TAG_AlgorithmVersion	Algorithm Version.
DCM_TAG_AlgorithmParameters	Algorithm Parameters.
DCM_TAG_FacetSequence	Facet Sequence.
DCM_TAG_SurfaceProcessingAlgorithmIdentificationSequence	Surface Processing Algorithm Identification Sequence.
DCM_TAG_AlgorithmName	Algorithm Name.
DCM_TAG_Group0070Length	Group 0070 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_GraphicAnnotationSequence	Graphic Annotation Sequence.
DCM_TAG_GraphicLayer	Graphic Layer.
DCM_TAG_BoundingBoxAnnotationUnits	Bounding Box Annotation Units.
DCM_TAG_AnchorPointAnnotationUnits	Anchor Point Annotation Units.
DCM_TAG_GraphicAnnotationUnits	Graphic Annotation Units.
DCM_TAG_UnformattedTextValue	Unformatted Text Value.
DCM_TAG_TextObjectSequence	Text Object Sequence.
DCM_TAG_GraphicObjectSequence	Graphic Object Sequence.
DCM_TAG_BoundingBoxTopLeftHandCorner	Bounding Box Top Left Hand Corner.
DCM_TAG_BoundingBoxBottomRightHandCorner	Bounding Box Bottom Right Hand

	Corner.
DCM_TAG_BoundingBoxTextHorizontalJustification	Bounding Box Text Horizontal Justification.
DCM_TAG_AnchorPoint	Anchor Point.
DCM_TAG_AnchorPointVisibility	Anchor Point Visibility.
DCM_TAG_GraphicDimensions	Graphic Dimensions.
DCM_TAG_NumberOfGraphicPoints	Number of Graphic Points.
DCM_TAG_GraphicData	Graphic Data.
DCM_TAG_GraphicType	Graphic Type.
DCM_TAG_GraphicFilled	Graphic Filled.
DCM_TAG_ImageRotationRetired	Image Rotation (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageHorizontalFlip	Image Horizontal Flip.
DCM_TAG_ImageRotation	Image Rotation.
DCM_TAG_DisplayedAreaTopLeftHandCornerTrial	Displayed Area Top Left Hand Corner (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DisplayedAreaBottomRightHandCornerTrial	Displayed Area Bottom Right Hand Corner (Trial). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DisplayedAreaTopLeftHandCorner	Displayed Area Top Left Hand Corner.
DCM_TAG_DisplayedAreaBottomRightHandCorner	Displayed Area Bottom Right Hand Corner.
DCM_TAG_DisplayedAreaSelectionSequence	Displayed Area Selection Sequence.
DCM_TAG_GraphicLayerSequence	Graphic Layer Sequence.
DCM_TAG_GraphicLayerOrder	Graphic Layer Order.
DCM_TAG_GraphicLayerRecommendedDisplayGrayscaleValue	Graphic Layer Recommended Display Grayscale Value.
DCM_TAG_GraphicLayerRecommendedDisplayRGBValue	Graphic Layer Recommended Display RGB Value. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_GraphicLayerDescription	Graphic Layer Description.
DCM_TAG_ContentLabel	Content Label.
DCM_TAG_PresentationLabel	Content Label. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ContentDescription	Content Description.
DCM_TAG_PresentationDescription	Content Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PresentationCreationDate	Presentation Creation Date.

DCM_TAG_PresentationCreationTime	Presentation Creation Time.
DCM_TAG_ContentCreatorName	Content Creator's Name.
DCM_TAG_ContentCreatorsName	Content Creator's Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PresentationCreatorsName	Content Creator's Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ContentCreatorIdentificationCodeSequence	Content Creator's Identification Code Sequence.
DCM_TAG_ContentCreatorsIdentificationCodeSequence	Content Creator's Identification Code Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_AlternateContentDescriptionSequence	Alternate Content Description Sequence.
DCM_TAG_PresentationSizeMode	Presentation Size Mode.
DCM_TAG_PresentationPixelSpacing	Presentation Pixel Spacing.
DCM_TAG_PresentationPixelAspectRatio	Presentation Pixel Aspect Ratio.
DCM_TAG_PresentationPixelMagnificationRatio	Presentation Pixel Magnification Ratio.
DCM_TAG_ShapeType	Shape Type.
DCM_TAG_RegistrationSequence	Registration Sequence.
DCM_TAG_MatrixRegistrationSequence	Matrix Registration Sequence.
DCM_TAG_MatrixSequence	Matrix Sequence.
DCM_TAG_FrameofReferenceTransformationMatrixType	Frame of Reference Transformation Matrix Type.
DCM_TAG_RegistrationTypeCodeSequence	Registration Type Code Sequence.
DCM_TAG_FiducialDescription	Fiducial Description.
DCM_TAG_FiducialIdentifier	Fiducial Identifier.
DCM_TAG_FiducialIdentifierCodeSequence	Fiducial Identifier Code Sequence.
DCM_TAG_ContourUncertaintyRadius	Contour Uncertainty Radius.
DCM_TAG_UsedFiducialsSequence	Used Fiducials Sequence.
DCM_TAG_GraphicCoordinatesDataSequence	Graphic Coordinates Data Sequence.
DCM_TAG_FiducialUID	Fiducial UID.
DCM_TAG_FiducialSetSequence	Fiducial Set Sequence.
DCM_TAG_FiducialSequence	Fiducial Sequence.
DCM_TAG_GraphicLayerRecommendedDisplayCIELabValue	Graphic Layer Recommended Display CIELab Value.
DCM_TAG_BlendingSequence	Blending Sequence.
DCM_TAG_RelativeOpacity	Relative Opacity.
DCM_TAG_ReferencedSpatialRegistrationSequence	Referenced Spatial Registration Sequence.

DCM_TAG_BlendingPosition	Blending Position.
DCM_TAG_Group0072Length	Group 0072 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_HangingProtocolName	Hanging Protocol Name.
DCM_TAG_HangingProtocolDescription	Hanging Protocol Description.
DCM_TAG_HangingProtocolLevel	Hanging Protocol Level.
DCM_TAG_HangingProtocolCreator	Hanging Protocol Creator.
DCM_TAG_HangingProtocolCreationDatetime	Hanging Protocol Creation DateTime. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_HangingProtocolDefinitionSequence	Hanging Protocol Definition Sequence.
DCM_TAG_HangingProtocolUserIdentificationCodeSequence	Hanging Protocol User Identification Code Sequence.
DCM_TAG_HangingProtocolUserGroupName	Hanging Protocol User Group Name.
DCM_TAG_SourceHangingProtocolSequence	Source Hanging Protocol Sequence.
DCM_TAG_NumberofPriorsReferenced	Number of Priors Referenced.
DCM_TAG_ImageSetsSequence	Image Sets Sequence.
DCM_TAG_ImageSetSelectorSequence	Image Set Selector Sequence.
DCM_TAG_ImageSetSelectorUsageFlag	Image Set Selector Usage Flag.
DCM_TAG_SelectorAttribute	Selector Attribute.
DCM_TAG_SelectorValueNumber	Selector Value Number.
DCM_TAG_TimeBasedImageSetsSequence	Time Based Image Sets Sequence.
DCM_TAG_ImageSetNumber	Image Set Number.
DCM_TAG_ImageSetSelectorCategory	Image Set Selector Category.
DCM_TAG_RelativeTime	Relative Time.
DCM_TAG_RelativeTimeUnits	Relative Time Units.
DCM_TAG_AbstractPriorValue	Abstract Prior Value.
DCM_TAG_AbstractPriorCodeSequence	Abstract Prior Code Sequence.
DCM_TAG_ImageSetLabel	Image Set Label.
DCM_TAG_SelectorAttributeVR	Selector Attribute VR.
DCM_TAG_SelectorSequencePointer	Selector Sequence Pointer.
DCM_TAG_SelectorSequencePointerPrivateCreator	Selector Sequence Pointer Private Creator.
DCM_TAG_SelectorAttributePrivateCreator	Selector Attribute Private Creator.
DCM_TAG_SelectorATValue	Selector AT Value.
DCM_TAG_SelectorCSValue	Selector CS Value.
DCM_TAG_SelectorISValue	Selector IS Value.
DCM_TAG_SelectorLOValue	Selector LO Value.
DCM_TAG_SelectorLTValue	Selector LT Value.
DCM_TAG_SelectorPNValue	Selector PN Value.
DCM_TAG_SelectorSHValue	Selector SH Value.
DCM_TAG_SelectorSTValue	Selector ST Value.

DCM_TAG_SelectorUTValue	Selector UT Value.
DCM_TAG_SelectorDSValue	Selector DS Value.
DCM_TAG_SelectorFDValue	Selector FD Value.
DCM_TAG_SelectorFLValue	Selector FL Value.
DCM_TAG_SelectorULValue	Selector UL Value.
DCM_TAG_SelectorUSValue	Selector US Value.
DCM_TAG_SelectorSLValue	Selector SL Value.
DCM_TAG_SelectorSSValue	Selector SS Value.
DCM_TAG_SelectorCodeSequenceValue	Selector Code Sequence Value.
DCM_TAG_NumberofScreens	Number of Screens.
DCM_TAG_NominalScreenDefinitionSequence	Nominal Screen Definition Sequence.
DCM_TAG_NumberofVerticalPixels	Number of Vertical Pixels.
DCM_TAG_NumberofHorizontalPixels	Number of Horizontal Pixels.
DCM_TAG_DisplayEnvironmentSpatialPosition	Display Environment Spatial Position.
DCM_TAG_ScreenMinimumGrayscaleBitDepth	Screen Minimum Grayscale Bit Depth.
DCM_TAG_ScreenMinimumColorBitDepth	Screen Minimum Color Bit Depth.
DCM_TAG_ApplicationMaximumRepaintTime	Application Maximum Repaint Time.
DCM_TAG_DisplaySetsSequence	Display Sets Sequence.
DCM_TAG_DisplaySetNumber	Display Set Number.
DCM_TAG_DisplaySetLabel	Display Set Label.
DCM_TAG_DisplaySetPresentationGroup	Display Set Presentation Group.
DCM_TAG_DisplaySetPresentationGroupDescription	Display Set Presentation Group Description.
DCM_TAG_PartialDataDisplayHandling	Partial Data Display Handling.
DCM_TAG_SynchronizedScrollingSequence	Synchronized Scrolling Sequence.
DCM_TAG_DisplaySetScrollingGroup	Display Set Scrolling Group.
DCM_TAG_NavigationIndicatorSequence	Navigation Indicator Sequence.
DCM_TAG_NavigationDisplaySet	Navigation Display Set.
DCM_TAG_ReferenceDisplaySets	Reference Display Sets.
DCM_TAG_ImageBoxesSequence	Image Boxes Sequence.
DCM_TAG_ImageBoxNumber	Image Box Number.
DCM_TAG_ImageBoxLayoutType	Image Box Layout Type.
DCM_TAG_ImageBoxTileHorizontalDimension	Image Box Tile Horizontal Dimension.
DCM_TAG_ImageBoxTileVerticalDimension	Image Box Tile Vertical Dimension.
DCM_TAG_ImageBoxScrollDirection	Image Box Scroll Direction.
DCM_TAG_ImageBoxSmallScrollType	Image Box Small Scroll Type.
DCM_TAG_ImageBoxSmallScrollAmount	Image Box Small Scroll Amount.
DCM_TAG_ImageBoxLargeScrollType	Image Box Large Scroll Type.
DCM_TAG_ImageBoxLargeScrollAmount	Image Box Large Scroll Amount.
DCM_TAG_ImageBoxOverlapPriority	Image Box Overlap Priority.
DCM_TAG_CineRelativeToRealTime	Cine Relative to Real-Time.
DCM_TAG_FilterOperationsSequence	Filter Operations Sequence.
DCM_TAG_FilterbyCategory	Filter-by Category.

DCM_TAG_FilterbyAttributePresence	Filter-by Attribute Presence.
DCM_TAG_FilterbyOperator	Filter-by Operator.
DCM_TAG_StructuredDisplayBackgroundCIELabValue	Structured Display Background CIELab Value.
DCM_TAG_EmptyImageBoxCIELabValue	Empty Image Box CIELab Value.
DCM_TAG_StructuredDisplayImageBoxSequence	Structured Display Image Box Sequence.
DCM_TAG_StructuredDisplayTextBoxSequence	Structured Display Text Box Sequence.
DCM_TAG_ReferencedFirstFrameSequence	Referenced First Frame Sequence.
DCM_TAG_ImageBoxSynchronizationSequence	Image Box Synchronization Sequence.
DCM_TAG_SynchronizedImageBoxList	Synchronized Image Box List.
DCM_TAG_TypeOfSynchronization	Type of Synchronization.
DCM_TAG_BlendingOperationType	Blending Operation Type.
DCM_TAG_ReformattingOperationType	Reformatting Operation Type.
DCM_TAG_ReformattingThickness	Reformatting Thickness.
DCM_TAG_ReformattingInterval	Reformatting Interval.
DCM_TAG_ReformattingOperationInitialViewDirection	Reformatting Operation Initial View Direction.
DCM_TAG_ThreeDRenderingType	3D Rendering Type.
DCM_TAG_RenderingType3D	3D Rendering Type. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_3DRenderingType	3D Rendering Type. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SortingOperationsSequence	Sorting Operations Sequence.
DCM_TAG_SortbyCategory	Sort-by Category.
DCM_TAG_SortingDirection	Sorting Direction.
DCM_TAG_DisplaySetPatientOrientation	Display Set Patient Orientation.
DCM_TAG_DisplaySetPatientOrientationCS2	Display Set Patient Orientation. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_VOIType	VOI (Value Of Interest) Type.
DCM_TAG_PseudocolorType	Pseudo-color Type.
DCM_TAG_ShowGrayscaleInverted	Show Grayscale Inverted.
DCM_TAG_ShowImageTrueSizeFlag	Show Image True Size Flag.
DCM_TAG_ShowGraphicAnnotationFlag	Show Graphic Annotation Flag.
DCM_TAG_ShowPatientDemographicsFlag	Show Patient Demographics Flag.
DCM_TAG_ShowAcquisitionTechniquesFlag	Show Acquisition Techniques Flag.
DCM_TAG_DisplaySetHorizontalJustification	Display Set Horizontal Justification.

DCM_TAG_DisplaySetVerticalJustification	Display Set Vertical Justification.
DCM_TAG_Group0074Length	Group 0074 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_UnifiedProcedureStepState	Unified Procedure Step State.
DCM_TAG_UnifiedProcedureStepProgressInformationSequence	Unified Procedure Step Progress Information Sequence.
DCM_TAG_UPSProgressInformationSequence	Unified Procedure Step Progress Information Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_UnifiedProcedureStepProgress	Unified Procedure Step Progress.
DCM_TAG_UnifiedProcedureStepProgressDescription	Unified Procedure Step Progress Description.
DCM_TAG_UnifiedProcedureStepCommunicationsURISequence	Unified Procedure Step Communications URI Sequence.
DCM_TAG_ContactURI	Contact URI.
DCM_TAG_ContactDisplayName	Contact Display Name.
DCM_TAG_UnifiedProcedureStepDiscontinuationReasonCodeSequence	Unified Procedure Step Discontinuation Reason Code Sequence.
DCM_TAG_BeamTaskSequence	Beam Task Sequence.
DCM_TAG_BeamTaskType	Beam Task Type.
DCM_TAG_BeamOrderIndex	Beam Order Index.
DCM_TAG_DeliveryVerificationImageSequence	Delivery Verification Image Sequence.
DCM_TAG_VerificationImageTiming	Verification Image Timing.
DCM_TAG_DoubleExposureFlag	Double Exposure Flag.
DCM_TAG_DoubleExposureOrdering	Double Exposure Ordering.
DCM_TAG_DoubleExposureMeterset	Double Exposure Meterset.
DCM_TAG_DoubleExposureFieldDelta	Double Exposure Field Delta.
DCM_TAG_RelatedReferenceRTImageSequence	Related Reference RT Image Sequence.
DCM_TAG_GeneralMachineVerificationSequence	General Machine Verification Sequence.
DCM_TAG_ConventionalMachineVerificationSequence	Conventional Machine Verification Sequence.
DCM_TAG_IonMachineVerificationSequence	Ion Machine Verification Sequence.
DCM_TAG_FailedAttributesSequence	Failed Attributes Sequence.
DCM_TAG_OverriddenAttributesSequence	Overridden Attributes Sequence.
DCM_TAG_ConventionalControlPointVerificationSequence	Conventional Control Point Verification Sequence.
DCM_TAG_IonControlPointVerificationSequence	Ion Control Point Verification Sequence.
DCM_TAG_AttributeOccurrenceSequence	Attribute Occurrence Sequence.
DCM_TAG_AttributeOccurrencePointer	Attribute Occurrence Pointer.
DCM_TAG_AttributeItemSelector	Attribute Item Selector.
DCM_TAG_AttributeOccurrencePrivateCreator	Attribute Occurrence Private Creator.

DCM_TAG_ScheduledProcedureStepPriority	Scheduled Procedure Step Priority.
DCM_TAG_WorklistLabel	Worklist Label.
DCM_TAG_ProcedureStepLabel	Procedure Step Label.
DCM_TAG_ScheduledProcessingParametersSequence	Scheduled Processing Parameters Sequence.
DCM_TAG_PerformedProcessingParametersSequence	Performed Processing Parameters Sequence.
DCM_TAG_UnifiedProcedureStepPerformedProcedureSequence	Unified Procedure Step Performed Procedure Sequence.
DCM_TAG_UPSPerformedProcedureSequence	Unified Procedure Step Performed Procedure Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RelatedProcedureStepSequence	Related Procedure Step Sequence.
DCM_TAG_ProcedureStepRelationshipType	Procedure Step Relationship Type.
DCM_TAG_DeletionLock	Deletion Lock.
DCM_TAG_ReceivingAE	Receiving AE.
DCM_TAG_RequestingAE	Requesting AE.
DCM_TAG_ReasonForCancellation	Reason for Cancellation.
DCM_TAG_SCPStatus	SCP Status.
DCM_TAG_SubscriptionListStatus	Subscription List Status.
DCM_TAG_UnifiedProcedureStepListStatus	Unified Procedure Step List Status.
DCM_TAG_UPSListStatus	Unified Procedure Step List Status. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_Group0088Length	Group 0088 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StorageMediaFilesetID	Storage Media File-set ID.
DCM_TAG_StorageMediaFilesetUID	Storage Media File-set UID.
DCM_TAG_IconImageSequence	Icon Image Sequence.
DCM_TAG_TopicTitle	Topic Title. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TopicSubject	Topic Subject. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TopicAuthor	Topic Author. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TopicKeywords	Topic Keywords. This tag is marked as retired in DICOM

	specification. See DICOM specification for alternatives.
DCM_TAG_Group0100Length	Group 0100 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SOPInstanceStatus	SOP Instance Status.
DCM_TAG_SOPAuthorizationDateTime	SOP Authorization Date Time.
DCM_TAG_SOPAuthorizationDateandTime	SOP Authorization Date and Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SOPAuthorizationComment	SOP Authorization Comment.
DCM_TAG_AuthorizationEquipmentCertificationNumber	Authorization Equipment Certification Number.
DCM_TAG_Group0400Length	Group 0400 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MacIDNumber	MAC ID Number.
DCM_TAG_MacCalculationTransferSyntaxUID	MAC Calculation Transfer Syntax UID.
DCM_TAG_MacCalcTransferSyntaxUID	MAC Calculation Transfer Syntax UID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MacAlgorithm	MAC Algorithm.
DCM_TAG_DataElementsSigned	Data Elements Signed.
DCM_TAG_DigitalSignatureUID	Digital Signature UID.
DCM_TAG_DigitalSignatureDateTime	Digital Signature DateTime.
DCM_TAG_CertificateType	Certificate Type.
DCM_TAG_CertificateOfSigner	Certificate of Signer.
DCM_TAG_Signature	The Signature.
DCM_TAG_CertifiedTimestampType	Certified Timestamp Type.
DCM_TAG_CertifiedTimestamp	Certified Timestamp.
DCM_TAG_DigitalSignaturePurposeCodeSequence	Digital Signature Purpose Code Sequence.
DCM_TAG_ReferencedDigitalSignatureSequence	Referenced Digital Signature Sequence.
DCM_TAG_ReferencedSOPInstanceMACSequence	Referenced SOP Instance MAC Sequence.
DCM_TAG_MAC	MAC (Message Authentication Code).
DCM_TAG_EncryptedAttributesSequence	Encrypted Attributes Sequence.
DCM_TAG_EncryptedContentTransferSyntaxUID	Encrypted Content Transfer Syntax UID.
DCM_TAG_EncryptedContent	Encrypted Content.
DCM_TAG_ModifiedAttributesSequence	Modified Attributes Sequence.
DCM_TAG_OriginalAttributesSequence	Original Attributes Sequence.
DCM_TAG_AttributeModificationDatetime	Attribute Modification DateTime.

	This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ModifyingSystem	Modifying System.
DCM_TAG_SourceOfPreviousValues	Source of Previous Values.
DCM_TAG_ReasonForTheAttributeModification	Reason for the Attribute Modification.
DCM_TAG_Group1000Length	Group 1000 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_EscapeTriplet	Escape Triplet. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RunLengthTriplet	Run Length Triplet. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_HuffmanTableSize	Huffman Table Size. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_HuffmanTableTriplet	Huffman Table Triplet. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ShiftTableSize	Shift Table Size. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ShiftTableTriplet	Shift Table Triplet. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group1010Length	Group 1010 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ZonalMap	Zonal Map. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group2000Length	Group 2000 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfCopies	Number of Copies.
DCM_TAG_PrinterConfigurationSequence	Printer Configuration Sequence.
DCM_TAG_PrinterConfigSequence	Printer Configuration Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PrintPriority	Print Priority.
DCM_TAG_MediumType	Medium Type.

DCM_TAG_FilmDestination	Film Destination.
DCM_TAG_FilmSessionLabel	Film Session Label.
DCM_TAG_MemoryAllocation	Memory Allocation.
DCM_TAG_MaximumMemoryAllocation	Maximum Memory Allocation.
DCM_TAG_MaxMemoryAllocation	Maximum Memory Allocation. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ColorImagePrintingFlag	Color Image Printing Flag. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CollationFlag	Collation Flag. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnnotationFlag	Annotation Flag. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageOverlayFlag	Image Overlay Flag. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PresentationLutFlag	Presentation LUT Flag. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageBoxPresentationLUTFlag	Image Box Presentation LUT Flag. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MemoryBitDepth	Memory Bit Depth.
DCM_TAG_PrintingBitDepth	Printing Bit Depth.
DCM_TAG_MediaInstalledSequence	Media Installed Sequence.
DCM_TAG_OtherMediaAvailableSequence	Other Media Available Sequence.
DCM_TAG_SupportedImageDisplayFormatsSequence	Supported Image Display Formats Sequence.
DCM_TAG_ReferencedFilmBoxSequence	Referenced Film Box Sequence.
DCM_TAG_ReferencedStoredPrintSequence	Referenced Stored Print Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group2010Length	Group 2010 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageDisplayFormat	Image Display Format.
DCM_TAG_AnnotationDisplayFormatID	Annotation Display Format ID.
DCM_TAG_FilmOrientation	Film Orientation.
DCM_TAG_FilmSizeID	Film Size ID.
DCM_TAG_PrinterResolutionID	Printer Resolution ID.
DCM_TAG_DefaultPrinterResolutionID	Default Printer Resolution ID.

DCM_TAG_MagnificationType	Magnification Type.
DCM_TAG_SmoothingType	Smoothing Type.
DCM_TAG_DefaultMagnificationType	Default Magnification Type.
DCM_TAG_OtherMagnificationTypesAvailable	Other Magnification Types Available.
DCM_TAG_DefaultSmoothingType	Default Smoothing Type.
DCM_TAG_OtherSmoothingTypesAvailable	Other Smoothing Types Available.
DCM_TAG_BorderDensity	Border Density.
DCM_TAG_EmptyImageDensity	Empty Image Density.
DCM_TAG_MinDensity	Min Density.
DCM_TAG_MaxDensity	Max Density.
DCM_TAG_Trim	Trim value.
DCM_TAG_ConfigurationInformation	Configuration Information.
DCM_TAG_ConfigurationInformationDescription	Configuration Information Description.
DCM_TAG_MaximumCollatedFilms	Maximum Collated Films.
DCM_TAG_Illumination	Illumination value.
DCM_TAG_ReflectedAmbientLight	Reflected Ambient Light.
DCM_TAG_PrinterPixelSpacing	Printer Pixel Spacing.
DCM_TAG_ReferencedFilmSessionSequence	Referenced Film Session Sequence.
DCM_TAG_ReferencedImageBoxSequence	Referenced Image Box Sequence.
DCM_TAG_ReferencedBasicImageBoxSequence	Referenced Image Box Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedBasicAnnotationBoxSequence	Referenced Basic Annotation Box Sequence.
DCM_TAG_Group2020Length	Group 2020 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageBoxPosition	Image Box Position.
DCM_TAG_FilmImagePosition	Image Box Position. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_Polarity	Image Polarity.
DCM_TAG_RequestedImageSize	Requested Image Size.
DCM_TAG_RequestedDecimateCropBehavior	Requested Decimate/Crop Behavior.
DCM_TAG_DecimateCropRequested	Requested Decimate/Crop Behavior. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RequestedResolutionID	Requested Resolution ID.

DCM_TAG_ResolutionIDRequested	Requested Resolution ID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RequestedImageSizeFlag	Requested Image Size Flag.
DCM_TAG_ImageSizeFlagRequested	Requested Image Size Flag. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DecimateCropResult	Decimate/Crop Result.
DCM_TAG_BasicGrayscaleImageSequence	Basic Grayscale Image Sequence.
DCM_TAG_PreformattedGrayscaleImageSequence	Basic Grayscale Image Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BasicColorImageSequence	Basic Color Image Sequence.
DCM_TAG_PreformattedColorImageSequence	Basic Color Image Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedImageOverlayBoxSequence	Referenced Image Overlay Box Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedVOILUTBoxSequence	Referenced VOI LUT Box Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group2030Length	Group 2030 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnnotationPosition	Annotation Position.
DCM_TAG_TextString	Text String.
DCM_TAG_Group2040Length	Group 2040 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedOverlayPlaneSequence	Referenced Overlay Plane Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedOverlayPlaneGroups	Referenced Overlay Plane Groups. This tag is marked as retired in DICOM specification. See DICOM specification for

	alternatives.
DCM_TAG_OverlayPixelDataSequence	Overlay Pixel Data Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayMagnificationType	Overlay Magnification Type. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlaySmoothingType	Overlay Smoothing Type. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayorImageMagnification	Overlay or Image Magnification. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MagnifytoNumberofColumns	Magnify to Number of Columns. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayForegroundDensity	Overlay Foreground Density. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayBackgroundDensity	Overlay Background Density. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayMode	Overlay Mode. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ThresholdDensity	Threshold Density. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedImageBoxSequenceRetired	Referenced Image Box Sequence (Retired). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group2050Length	Group 2050 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PresentationLutSequence	Presentation LUT Sequence.
DCM_TAG_PresentationLutShape	Presentation LUT Shape.
DCM_TAG_ReferencedPresentationLutSequence	Referenced Presentation LUT Sequence.
DCM_TAG_Group2100Length	Group 2100 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PrintJobID	Print Job ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ExecutionStatus	Execution Status.
DCM_TAG_ExecutionStatusInfo	Execution Status Info.

DCM_TAG_CreationDate	Creation Date.
DCM_TAG_CreationTime	Creation Time.
DCM_TAG_Originator	The Originator.
DCM_TAG_DestinationAe	Destination AE. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OwnerID	Owner Identifier.
DCM_TAG_NumberOfFilms	Number of Films.
DCM_TAG_ReferencedPrintJobSequencePullStoredPrint	Referenced Print Job Sequence (Pull Stored Print). This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group2110Length	Group 2110 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PrinterStatus	Printer Status.
DCM_TAG_PrinterStatusInfo	Printer Status Info.
DCM_TAG_PrinterName	Printer Name.
DCM_TAG_PrintQueueID	Print Queue ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group2120Length	Group 2120 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_QueueStatus	Queue Status. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PrintJobDescriptionSequence	Print Job Description Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedPrintJobSequence	Referenced Print Job Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group2130Length	Group 2130 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PrintManagementCapabilitiesSequence	Print Management Capabilities Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PrinterCharacteristicsSequence	Printer Characteristics Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_FilmBoxContentSequence	Film Box Content Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageBoxContentSequence	Image Box Content Sequence. This tag is marked as retired in

	DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AnnotationContentSequence	Annotation Content Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ImageOverlayBoxContentSequence	Image Overlay Box Content Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PresentationLutContentSequence	Presentation LUT Content Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ProposedStudySequence	Proposed Study Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OriginalImageSequence	Original Image Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group2200Length	Group 2200 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_LabelUsingInformationExtractedFromInstances	Label Using Information Extracted From Instances.
DCM_TAG_LabelText	Label Text.
DCM_TAG_LabelStyleSelection	Label Style Selection.
DCM_TAG_MediaDisposition	Media Disposition.
DCM_TAG_BarcodeValue	Barcode Value.
DCM_TAG_BarcodeSymbology	Barcode Symbology.
DCM_TAG_AllowMediaSplitting	Allow Media Splitting.
DCM_TAG_IncludeNonDICOMObjects	Include Non-DICOM Objects.
DCM_TAG_IncludeDisplayApplication	Include Display Application.
DCM_TAG_PreserveCompositeInstancesAfterMediaCreation	Preserve Composite Instances After Media Creation.
DCM_TAG_TotalNumberofPiecesofMediaCreated	Total Number of Pieces of Media Created.
DCM_TAG_RequestedMediaApplicationProfile	Requested Media Application Profile.
DCM_TAG_ReferencedStorageMediaSequence	Referenced Storage Media Sequence.
DCM_TAG_FailureAttributes	Failure Attributes.
DCM_TAG_AllowLossyCompression	Allow Lossy Compression.
DCM_TAG_RequestPriority	Request Priority.
DCM_TAG_Group3002Length	Group 3002 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RtImageLabel	RT Image Label.
DCM_TAG_RtImageName	RT Image Name.
DCM_TAG_RtImageDescription	RT Image Description.
DCM_TAG_ReportedValuesOrigin	Reported Values Origin.

DCM_TAG_RtImagePlane	RT Image Plane.
DCM_TAG_XRayImageReceptorTranslation	X-Ray Image Receptor Translation.
DCM_TAG_XrayImageReceptorAngle	X-Ray Image Receptor Angle.
DCM_TAG_RtImageOrientation	RT Image Orientation.
DCM_TAG_ImagePlanePixelSpacing	Image Plane Pixel Spacing.
DCM_TAG_RtImagePosition	RT Image Position.
DCM_TAG_RadiationMachineName	Radiation Machine Name.
DCM_TAG_RadiationMachineSad	Radiation Machine SAD.
DCM_TAG_RadiationMachineSsd	Radiation Machine SSD.
DCM_TAG_RtImageSid	RT Image SID.
DCM_TAG_SourceToReferenceObjectDistance	Source to Reference Object Distance.
DCM_TAG_FractionNumber	Fraction Number.
DCM_TAG_ExposureSequence	Exposure Sequence.
DCM_TAG_MetersetExposure	Meterset Exposure.
DCM_TAG_DiaphragmPosition	Diaphragm Position.
DCM_TAG_FluenceMapSequence	Fluence Map Sequence.
DCM_TAG_FluenceDataSource	Fluence Data Source.
DCM_TAG_FluenceDataScale	Fluence Data Scale.
DCM_TAG_PrimaryFluenceModeSequence	Primary Fluence Mode Sequence.
DCM_TAG_FluenceMode	Fluence Mode.
DCM_TAG_FluenceModeID	Fluence Mode ID.
DCM_TAG_Group3004Length	Group 3004 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DvhType	DVH (Dose-volume histogram) Type.
DCM_TAG_DoseUnits	Dose Units.
DCM_TAG_DoseUnit	Dose Units. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DoseType	Dose Type.
DCM_TAG_DoseComment	Dose Comment.
DCM_TAG_NormalizationPoint	Normalization Point.
DCM_TAG_DoseSummationType	Dose Summation Type.
DCM_TAG_GridFrameOffsetVector	Grid Frame Offset Vector.
DCM_TAG_DoseGridScaling	Dose Grid Scaling.
DCM_TAG_RtDoseRoiSequence	RT Dose ROI Sequence.
DCM_TAG_DoseValue	Dose Value.
DCM_TAG_TissueHeterogeneityCorrection	Tissue Heterogeneity Correction.
DCM_TAG_DvhNormalizationPoint	DVH Normalization Point.
DCM_TAG_DvhNormalizationDoseValue	DVH Normalization Dose Value.
DCM_TAG_DvhSequence	DVH Sequence.
DCM_TAG_DvhDoseScaling	DVH Dose Scaling.
DCM_TAG_DvhVolumeUnits	DVH Volume Units.
DCM_TAG_DvhVolumeUnit	DVH Volume Units. This tag

	name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DvhNumberOfBins	DVH Number of Bins.
DCM_TAG_DvhData	The DVH Data.
DCM_TAG_DvhReferencedRoiSequence	DVH Referenced ROI Sequence.
DCM_TAG_DvhRoiContributionType	DVH ROI Contribution Type.
DCM_TAG_DvhMinimumDose	DVH Minimum Dose.
DCM_TAG_DvhMaximumDose	DVH Maximum Dose.
DCM_TAG_DvhMeanDose	DVH Mean Dose.
DCM_TAG_Group3006Length	Group 3006 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_StructureSetLabel	Structure Set Label.
DCM_TAG_StructureSetName	Structure Set Name.
DCM_TAG_StructureSetDescription	Structure Set Description.
DCM_TAG_StructureSetDate	Structure Set Date.
DCM_TAG_StructureSetTime	Structure Set Time.
DCM_TAG_ReferencedFrameOfReferenceSequence	Referenced Frame of Reference Sequence.
DCM_TAG_RtReferencedStudySequence	RT Referenced Study Sequence.
DCM_TAG_RtReferencedSeriesSequence	RT Referenced Series Sequence.
DCM_TAG_ContourImageSequence	Contour Image Sequence.
DCM_TAG_StructureSetRoiSequence	Structure Set ROI Sequence.
DCM_TAG_RoiNumber	ROI Number.
DCM_TAG_ReferencedFrameOfReferenceUID	Referenced Frame of Reference UID.
DCM_TAG_RoiName	ROI (Region of Interest) Name.
DCM_TAG_RoiDescription	ROI Description.
DCM_TAG_RoiDisplayColor	ROI Display Color.
DCM_TAG_RoiVolume	ROI Volume.
DCM_TAG_RtRelatedRoiSequence	RT Related ROI Sequence.
DCM_TAG_RtRoiRelationship	RT ROI Relationship.
DCM_TAG_RoiGenerationAlgorithm	ROI Generation Algorithm.
DCM_TAG_RoiGenerationDescription	ROI Generation Description.
DCM_TAG_RoiContourSequence	ROI Contour Sequence.
DCM_TAG_ContourSequence	Contour Sequence.
DCM_TAG_ContourGeometricType	Contour Geometric Type.
DCM_TAG_ContourSlabThickness	Contour Slab Thickness.
DCM_TAG_ContourOffsetVector	Contour Offset Vector.
DCM_TAG_NumberOfContourPoints	Number of Contour Points.
DCM_TAG_ContourNumber	Contour Number.
DCM_TAG_AttachedContours	Attached Contours.
DCM_TAG_ContourData	Contour Data.
DCM_TAG_RtRoiObservationsSequence	RT ROI Observations Sequence.
DCM_TAG_RtRoiObservationSequence	RT ROI Observations Sequence. This tag name has been deprecated and will be removed

	from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ObservationNumber	Observation Number.
DCM_TAG_ReferencedRoiNumber	Referenced ROI Number.
DCM_TAG_RoiObservationLabel	ROI Observation Label.
DCM_TAG_RtRoiIdentificationCodeSequence	RT ROI Identification Code Sequence.
DCM_TAG_RoiObservationDescription	ROI Observation Description.
DCM_TAG_RelatedRtRoiObservationsSequence	Related RT ROI Observations Sequence.
DCM_TAG_RelatedRtRoiObservationSequence	Related RT ROI Observations Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RtRoiInterpretedType	RT ROI Interpreted Type.
DCM_TAG_RoiInterpreter	ROI Interpreter.
DCM_TAG_RoiPhysicalPropertiesSequence	ROI Physical Properties Sequence.
DCM_TAG_RoiPhysicalPropertySequence	ROI Physical Properties Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RoiPhysicalProperty	ROI Physical Property.
DCM_TAG_RoiPhysicalPropertyValue	ROI Physical Property Value.
DCM_TAG_ROIElementalCompositionSequence	ROI Elemental Composition Sequence.
DCM_TAG_ROIElementalCompositionAtomicNumber	ROI Elemental Composition Atomic Number.
DCM_TAG_ROIElementalCompositionAtomicMassFraction	ROI Elemental Composition Atomic Mass Fraction.
DCM_TAG_FrameOfReferenceRelationshipSequence	Frame of Reference Relationship Sequence.
DCM_TAG_RelatedFrameOfReferenceUID	Related Frame of Reference UID.
DCM_TAG_FrameOfReferenceTransformationType	Frame of Reference Transformation Type.
DCM_TAG_FrameOfReferenceTransformationMatrix	Frame of Reference Transformation Matrix.
DCM_TAG_FrameOfReferenceTransformationComment	Frame of Reference Transformation Comment.
DCM_TAG_Group3008Length	Group 3008 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MeasuredDoseReferenceSequence	Measured Dose Reference Sequence.
DCM_TAG_MeasuredDoseRefSequence	Measured Dose Reference Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the

	tag with the same value defined in the previous line.
DCM_TAG_MeasuredDoseDescription	Measured Dose Description.
DCM_TAG_MeasuredDoseDesc	Measured Dose Description. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MeasuredDoseType	Measured Dose Type.
DCM_TAG_MeasuredDoseValue	Measured Dose Value.
DCM_TAG_TreatmentSessionBeamSequence	Treatment Session Beam Sequence.
DCM_TAG_TreatmentSessionIonBeamSequence	Treatment Session Ion Beam Sequence.
DCM_TAG_CurrentFractionNumber	Current Fraction Number.
DCM_TAG_TreatmentControlPointDate	Treatment Control Point Date.
DCM_TAG_TreatmentControlPointTime	Treatment Control Point Time.
DCM_TAG_TreatmentTerminationStatus	Treatment Termination Status.
DCM_TAG_TreatmentTerminationCode	Treatment Termination Code.
DCM_TAG_TreatmentVerificationStatus	Treatment Verification Status.
DCM_TAG_ReferencedTreatmentRecordSequence	Referenced Treatment Record Sequence.
DCM_TAG_RefTreatmentRecSequence	Referenced Treatment Record Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SpecifiedPrimaryMeterset	Specified Primary Meterset.
DCM_TAG_SpecifiedSecondaryMeterset	Specified Secondary Meterset.
DCM_TAG_DeliveredPrimaryMeterset	Delivered Primary Meterset.
DCM_TAG_DeliveredSecondaryMeterset	Delivered Secondary Meterset.
DCM_TAG_SpecifiedTreatmentTime	Specified Treatment Time.
DCM_TAG_DeliveredTreatmentTime	Delivered Treatment Time.
DCM_TAG_ControlPointDeliverySequence	Control Point Delivery Sequence.
DCM_TAG_IonControlPointDeliverySequence	Ion Control Point Delivery Sequence.
DCM_TAG_SpecifiedMeterset	Specified Meterset.
DCM_TAG_DeliveredMeterset	Delivered Meterset.
DCM_TAG_MetersetRateSet	Meterset Rate Set.
DCM_TAG_MetersetRateDelivered	Meterset Rate Delivered.
DCM_TAG_ScanSpotMetersetsDelivered	Scan Spot Metersets Delivered.
DCM_TAG_DoseRateDelivered	Dose Rate Delivered.
DCM_TAG_TreatmentSummaryCalculatedDoseReferenceSequence	Treatment Summary Calculated Dose Reference Sequence.
DCM_TAG_TreatmentSummaryCalcDoseRef	Treatment Summary Calculated Dose Reference Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous

	line.
DCM_TAG_CumulativeDoseToDoseReference	Cumulative Dose to Dose Reference.
DCM_TAG_CumulativeDoseToDoseRef	Cumulative Dose to Dose Reference. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_FirstTreatmentDate	First Treatment Date.
DCM_TAG_MostRecentTreatmentDate	Most Recent Treatment Date.
DCM_TAG_NumberOfFractionsDelivered	Number of Fractions Delivered.
DCM_TAG_NumberOfFractionDelivered	Number of Fractions Delivered. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OverrideSequence	Override Sequence.
DCM_TAG_ParameterSequencePointer	Parameter Sequence Pointer.
DCM_TAG_OverrideParameterPointer	Override Parameter Pointer.
DCM_TAG_ParameterItemIndex	Parameter Item Index.
DCM_TAG_MeasuredDoseReferenceNumber	Measured Dose Reference Number.
DCM_TAG_MeasuredDoseRefNumber	Measured Dose Reference Number. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ParameterPointer	Parameter Pointer.
DCM_TAG_OverrideReason	Override Reason.
DCM_TAG_CorrectedParameterSequence	Corrected Parameter Sequence.
DCM_TAG_CorrectionValue	Correction Value.
DCM_TAG_CalculatedDoseReferenceSequence	Calculated Dose Reference Sequence.
DCM_TAG_CalcDoseRefSequence	Calculated Dose Reference Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CalculatedDoseReferenceNumber	Calculated Dose Reference Number.
DCM_TAG_CalcDoseRefNumber	Calculated Dose Reference Number. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CalculatedDoseReferenceDescription	Calculated Dose Reference Description.
DCM_TAG_CalcDoseRefDesc	Calculated Dose Reference Description. This tag name has

	been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CalculatedDoseReferenceDoseValue	Calculated Dose Reference Dose Value.
DCM_TAG_CalcDoseRefDoseValue	Calculated Dose Reference Dose Value. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_StartMeterset	Start Meterset.
DCM_TAG_EndMeterset	End Meterset.
DCM_TAG_ReferencedMeasuredDoseReferenceSequence	Referenced Measured Dose Reference Sequence.
DCM_TAG_RefMeasuredDoseRefSequence	Referenced Measured Dose Reference Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedMeasuredDoseReferenceNumber	Referenced Measured Dose Reference Number.
DCM_TAG_RefMeasuredDoseRefNumber	Referenced Measured Dose Reference Number. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedCalculatedDoseReferenceSequence	Referenced Calculated Dose Reference Sequence.
DCM_TAG_RefCalcDoseRefSequence	Referenced Calculated Dose Reference Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedCalculatedDoseReferenceNumber	Referenced Calculated Dose Reference Number.
DCM_TAG_RefCalcDoseRefNumber	Referenced Calculated Dose Reference Number. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BeamLimitingDeviceLeafPairsSequence	Beam Limiting Device Leaf Pairs Sequence.
DCM_TAG_BeamLimitingDeviceLeafPairSequence	Beam Limiting Device Leaf Pairs Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RecordedWedgeSequence	Recorded Wedge Sequence.

DCM_TAG_RecordedCompensatorSequence	Recorded Compensator Sequence.
DCM_TAG_RecordedBlockSequence	Recorded Block Sequence.
DCM_TAG_TreatmentSummaryMeasuredDoseReferenceSequence	Treatment Summary Measured Dose Reference Sequence.
DCM_TAG_TreatmentSummarySequence	Treatment Summary Measured Dose Reference Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RecordedSnoutSequence	Recorded Snout Sequence.
DCM_TAG_RecordedRangeShifterSequence	Recorded Range Shifter Sequence.
DCM_TAG_RecordedLateralSpreadingDeviceSequence	Recorded Lateral Spreading Device Sequence.
DCM_TAG_RecordedRangeModulatorSequence	Recorded Range Modulator Sequence.
DCM_TAG_RecordedSourceSequence	Recorded Source Sequence.
DCM_TAG_SourceSerialNumber	Source Serial Number.
DCM_TAG_TreatmentSessionApplicationSetupSequence	Treatment Session Application Setup Sequence.
DCM_TAG_TreatmentSessionSetupSequence	Treatment Session Application Setup Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ApplicationSetupCheck	Application Setup Check.
DCM_TAG_AppSetupCheck	Application Setup Check. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RecordedBrachyAccessoryDeviceSequence	Recorded Brachy Accessory Device Sequence.
DCM_TAG_RecordedBrachyAccDeviceSequence	Recorded Brachy Accessory Device Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedBrachyAccessoryDeviceNumber	Referenced Brachy Accessory Device Number.
DCM_TAG_RefBrachyAccDevNumber	Referenced Brachy Accessory Device Number. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RecordedChannelSequence	Recorded Channel Sequence.
DCM_TAG_SpecifiedChannelTotalTime	Specified Channel Total Time.
DCM_TAG_DeliveredChannelTotalTime	Delivered Channel Total Time.
DCM_TAG_SpecifiedNumberofPulses	Specified Number of Pulses.

DCM_TAG_DeliveredNumberofPulses	Delivered Number of Pulses.
DCM_TAG_SpecifiedPulseRepetitionInterval	Specified Pulse Repetition Interval.
DCM_TAG_DeliveredPulseRepetitionInterval	Delivered Pulse Repetition Interval.
DCM_TAG_RecordedSourceApplicatorSequence	Recorded Source Applicator Sequence.
DCM_TAG_ReferencedSourceApplicatorNumber	Referenced Source Applicator Number.
DCM_TAG_RecordedChannelShieldSequence	Recorded Channel Shield Sequence.
DCM_TAG_ReferencedChannelShieldNumber	Referenced Channel Shield Number.
DCM_TAG_BrachyControlPointDeliveredSequence	Brachy Control Point Delivered Sequence.
DCM_TAG_SafePositionExitDate	Safe Position Exit Date.
DCM_TAG_SafePositionExitTime	Safe Position Exit Time.
DCM_TAG_SafePositionReturnDate	Safe Position Return Date.
DCM_TAG_SafePositionReturnTime	Safe Position Return Time.
DCM_TAG_CurrentTreatmentStatus	Current Treatment Status.
DCM_TAG_TreatmentStatusComment	Treatment Status Comment.
DCM_TAG_FractionGroupSummarySequence	Fraction Group Summary Sequence.
DCM_TAG_ReferencedFractionNumber	Referenced Fraction Number.
DCM_TAG_FractionGroupType	Fraction Group Type.
DCM_TAG_BeamStopperPosition	Beam Stopper Position.
DCM_TAG_FractionStatusSummarySequence	Fraction Status Summary Sequence.
DCM_TAG_TreatmentDate	Treatment Date.
DCM_TAG_TreatmentTime	Treatment Time.
DCM_TAG_Group300ALength	Group 300A Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RtPlanLabel	RT Plan Label.
DCM_TAG_RtPlanName	RT Plan Name.
DCM_TAG_RtPlanDescription	RT Plan Description.
DCM_TAG_RtPlanDate	RT Plan Date.
DCM_TAG_RtPlanTime	RT Plan Time.
DCM_TAG_TreatmentProtocols	Treatment Protocols.
DCM_TAG_TreatmentProtocol	Treatment Protocols. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PlanIntent	Plan Intent.
DCM_TAG_TreatmentIntent	Plan Intent. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TreatmentSites	Treatment Sites.

DCM_TAG_TreatmentSite	Treatment Sites. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RtPlanGeometry	RT Plan Geometry.
DCM_TAG_PrescriptionDescription	Prescription Description.
DCM_TAG_DoseReferenceSequence	Dose Reference Sequence.
DCM_TAG_DoseReferenceNumber	Dose Reference Number.
DCM_TAG_DoseReferenceUID	Dose Reference UID.
DCM_TAG_DoseReferenceStructureType	Dose Reference Structure Type.
DCM_TAG_NominalBeamEnergyUnit	Nominal Beam Energy Unit.
DCM_TAG_DoseReferenceDescription	Dose Reference Description.
DCM_TAG_DoseReferencePointCoordinates	Dose Reference Point Coordinates.
DCM_TAG_DoseReferencePointCoordinate	Dose Reference Point Coordinates. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_NominalPriorDose	Nominal Prior Dose.
DCM_TAG_DoseReferenceType	Dose Reference Type.
DCM_TAG_ConstraintWeight	Constraint Weight.
DCM_TAG_ConsraintWeight	Constraint Weight. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_DeliveryWarningDose	Delivery Warning Dose.
DCM_TAG_DeliveryMaximumDose	Delivery Maximum Dose.
DCM_TAG_TargetMinimumDose	Target Minimum Dose.
DCM_TAG_TargetPrescriptionDose	Target Prescription Dose.
DCM_TAG_TargetMaximumDose	Target Maximum Dose.
DCM_TAG_TargetUnderdoseVolumeFraction	Target Underdose Volume Fraction.
DCM_TAG_OrganAtRiskFullVolumeDose	Organ at Risk Full-volume Dose.
DCM_TAG_OrganAtRiskLimitDose	Organ at Risk Limit Dose.
DCM_TAG_OrganAtRiskMaximumDose	Organ at Risk Maximum Dose.
DCM_TAG_OrganAtRiskOverdoseVolumeFraction	Organ at Risk Overdose Volume Fraction.
DCM_TAG_ToleranceTableSequence	Tolerance Table Sequence.
DCM_TAG_ToleranceTableNumber	Tolerance Table Number.
DCM_TAG_ToleranceTableLabel	Tolerance Table Label.
DCM_TAG_GantryAngleTolerance	Gantry Angle Tolerance.
DCM_TAG_BeamLimitingDeviceAngleTolerance	Beam Limiting Device Angle Tolerance.
DCM_TAG_BeamLimitingDeviceToleranceSequence	Beam Limiting Device Tolerance Sequence.
DCM_TAG_BeamLimitingDevicePositionTolerance	Beam Limiting Device Position Tolerance.

DCM_TAG_SnoutPositionTolerance	Snout Position Tolerance.
DCM_TAG_PatientSupportAngleTolerance	Patient Support Angle Tolerance.
DCM_TAG_TableTopEccentricAngleTolerance	Table Top Eccentric Angle Tolerance.
DCM_TAG_TableTopPitchAngleTolerance	Table Top Pitch Angle Tolerance.
DCM_TAG_TableTopRollAngleTolerance	Table Top Roll Angle Tolerance.
DCM_TAG_TableTopVerticalPositionTolerance	Table Top Vertical Position Tolerance.
DCM_TAG_TableTopLongitudinalPositionTolerance	Table Top Longitudinal Position Tolerance.
DCM_TAG_TableTopLogitudinalPositionTolerance	Table Top Longitudinal Position Tolerance. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TableTopLateralPositionTolerance	Table Top Lateral Position Tolerance.
DCM_TAG_RtPlanRelationship	RT Plan Relationship.
DCM_TAG_FractionGroupSequence	Fraction Group Sequence.
DCM_TAG_FractionGroupNumber	Fraction Group Number.
DCM_TAG_FractionGroupDescription	Fraction Group Description.
DCM_TAG_NumberOfFractionsPlanned	Number of Fractions Planned.
DCM_TAG_NumberOfFractionPlanned	Number of Fractions Planned. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_NumberOfFractionPatternDigitsPerDay	Number of Fraction Pattern Digits Per Day.
DCM_TAG_NumberOfFractionPerDay	Number of Fraction Pattern Digits Per Day. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RepeatFractionCycleLength	Repeat Fraction Cycle Length.
DCM_TAG_FractionPattern	Fraction Pattern.
DCM_TAG_NumberOfBeams	Number of Beams.
DCM_TAG_BeamDoseSpecificationPoint	Beam Dose Specification Point.
DCM_TAG_BeamDose	Beam Dose.
DCM_TAG_BeamMeterset	Beam Meterset.
DCM_TAG_BeamDosePointDepth	Beam Dose Point Depth.
DCM_TAG_BeamDosePointEquivalentDepth	Beam Dose Point Equivalent Depth.
DCM_TAG_BeamDosePointSSD	Beam Dose Point SSD.
DCM_TAG_NumberOfBrachyApplicationSetups	Number of Brachy Application Setups.
DCM_TAG_NumberOfBrachyApplicationSetup	Number of Brachy Application Setups. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with

	the same value defined in the previous line.
DCM_TAG_BrachyApplicationSetupDoseSpecificationPoint	Brachy Application Setup Dose Specification Point.
DCM_TAG_BrachySetupDoseSpecificationPoint	Brachy Application Setup Dose Specification Point. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BrachyApplicationSetupDose	Brachy Application Setup Dose.
DCM_TAG_BrachySetupDose	Brachy Application Setup Dose. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BeamSequence	Beam Sequence.
DCM_TAG_TreatmentMachineName	Treatment Machine Name.
DCM_TAG_PrimaryDosimeterUnit	Primary Dosimeter Unit.
DCM_TAG_SourceAxisDistance	Source-Axis Distance.
DCM_TAG_BeamLimitingDeviceSequence	Beam Limiting Device Sequence.
DCM_TAG_RtBeamLimitingDeviceType	RT Beam Limiting Device Type.
DCM_TAG_SourceToBeamLimitingDeviceDistance	Source to Beam Limiting Device Distance.
DCM_TAG_IsocentertoBeamLimitingDeviceDistance	Isocenter to Beam Limiting Device Distance.
DCM_TAG_NumberOfLeafJawPairs	Number of Leaf/Jaw Pairs.
DCM_TAG_LeafPositionBoundaries	Leaf Position Boundaries.
DCM_TAG_BeamNumber	Beam Number.
DCM_TAG_BeamName	Beam Name.
DCM_TAG_BeamDescription	Beam Description.
DCM_TAG_BeamType	Beam Type.
DCM_TAG_RadiationType	Radiation Type.
DCM_TAG_HighDoseTechniqueType	High-Dose Technique Type.
DCM_TAG_ReferenceImageNumber	Reference Image Number.
DCM_TAG_PlannedVerificationImageSequence	Planned Verification Image Sequence.
DCM_TAG_ImagingDeviceSpecificAcquisitionParameters	Imaging Device-Specific Acquisition Parameters.
DCM_TAG_ImagingDeviceSpecificAcquisitionParameter	Imaging Device-Specific Acquisition Parameters. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ImagingDeviceSpecificAcqParameter	Imaging Device-Specific Acquisition Parameters. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TreatmentDeliveryType	Treatment Delivery Type.

DCM_TAG_NumberOfWedges	Number of Wedges.
DCM_TAG_NumberOfWedge	Number of Wedges. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_WedgeSequence	Wedge Sequence.
DCM_TAG_WedgeNumber	Wedge Number.
DCM_TAG_WedgeType	Wedge Type.
DCM_TAG_WedgeID	Wedge Identifier.
DCM_TAG_WedgeAngle	Wedge Angle.
DCM_TAG_WedgeFactor	Wedge Factor.
DCM_TAG_TotalWedgeTrayWaterEquivalentThickness	Total Wedge Tray Water-Equivalent Thickness.
DCM_TAG_WedgeOrientation	Wedge Orientation.
DCM_TAG_IsocentertoWedgeTrayDistance	Isocenter to Wedge Tray Distance.
DCM_TAG_SourceToWedgeTrayDistance	Source to Wedge Tray Distance.
DCM_TAG_WedgeThinEdgePosition	Wedge Thin Edge Position.
DCM_TAG_BolusID	Bolus Identifier.
DCM_TAG_BolusDescription	Bolus Description.
DCM_TAG_NumberOfCompensators	Number of Compensators.
DCM_TAG_NumberOfCompensator	Number of Compensators. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_MaterialID	Material ID.
DCM_TAG_TotalCompensatorTrayFactor	Total Compensator Tray Factor.
DCM_TAG_CompensatorSequence	Compensator Sequence.
DCM_TAG_CompensatorNumber	Compensator Number.
DCM_TAG_CompensatorID	Compensator ID.
DCM_TAG_SourceToCompensatorTrayDistance	Source to Compensator Tray Distance.
DCM_TAG_CompensatorRows	Compensator Rows.
DCM_TAG_CompensatorRow	Compensator Rows. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CompensatorColumns	Compensator Columns.
DCM_TAG_CompensatorColumn	Compensator Columns. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_CompensatorPixelSpacing	Compensator Pixel Spacing.
DCM_TAG_CompensatorPosition	Compensator Position.
DCM_TAG_CompensatorTransmissionData	Compensator Transmission Data.
DCM_TAG_CompensatorThicknessData	Compensator Thickness Data.

DCM_TAG_NumberOfBoli	Number of Boli.
DCM_TAG_CompensatorType	Compensator Type.
DCM_TAG_NumberOfBlocks	Number of Blocks.
DCM_TAG_NumberOfBlock	Number of Blocks. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TotalBlockTrayFactor	Total Block Tray Factor.
DCM_TAG_TotalBlockTrayWaterEquivalentThickness	Total Block Tray Water-Equivalent Thickness.
DCM_TAG_BlockSequence	Block Sequence.
DCM_TAG_BlockTrayID	Block Tray ID.
DCM_TAG_SourceToBlockTrayDistance	Source to Block Tray Distance.
DCM_TAG_IsocentertoBlockTrayDistance	Isocenter to Block Tray Distance.
DCM_TAG_BlockType	Block Type.
DCM_TAG_AccessoryCode	Accessory Code.
DCM_TAG_BlockDivergence	Block Divergence.
DCM_TAG_BlockMountingPosition	Block Mounting Position.
DCM_TAG_BlockNumber	Block Number.
DCM_TAG_BlockName	Block Name.
DCM_TAG_BlockThickness	Block Thickness.
DCM_TAG_BlockTransmission	Block Transmission.
DCM_TAG_BlockNumberOfPoints	Block Number of Points.
DCM_TAG_BlockData	Block Data.
DCM_TAG_ApplicatorSequence	Applicator Sequence.
DCM_TAG_ApplicatorID	Applicator ID.
DCM_TAG_ApplicatorType	Applicator Type.
DCM_TAG_ApplicatorDescription	Applicator Description.
DCM_TAG_CumulativeDoseReferenceCoefficient	Cumulative Dose Reference Coefficient.
DCM_TAG_FinalCumulativeMetersetWeight	Final Cumulative Meterset Weight.
DCM_TAG_NumberOfControlPoints	Number of Control Points.
DCM_TAG_ControlPointSequence	Control Point Sequence.
DCM_TAG_ControlPointIndex	Control Point Index.
DCM_TAG_NominalBeamEnergy	Nominal Beam Energy.
DCM_TAG_DoseRateSet	Dose Rate Set.
DCM_TAG_WedgePositionSequence	Wedge Position Sequence.
DCM_TAG_WedgePosition	Wedge Position.
DCM_TAG_BeamLimitingDevicePositionSequence	Beam Limiting Device Position Sequence.
DCM_TAG_LeafJawPositions	Leaf/Jaw Positions.
DCM_TAG_LeafJawPosition	Leaf/Jaw Positions.
DCM_TAG_GantryAngle	Gantry Angle.
DCM_TAG_GantryRotationDirection	Gantry Rotation Direction.
DCM_TAG_BeamLimitingDeviceAngle	Beam Limiting Device Angle.
DCM_TAG_BeamLimitingDeviceRotationDirection	Beam Limiting Device Rotation Direction.
DCM_TAG_PatientSupportAngle	Patient Support Angle.

DCM_TAG_PatientSupportRotationDirection	Patient Support Rotation Direction.
DCM_TAG_TableTopEccentricAxisDistance	Table Top Eccentric Axis Distance.
DCM_TAG_TableTopEccentricAngle	Table Top Eccentric Angle.
DCM_TAG_TableTopEccentricRotationDirection	Table Top Eccentric Rotation Direction.
DCM_TAG_TableTopVerticalPosition	Table Top Vertical Position.
DCM_TAG_TableTopLongitudinalPosition	Table Top Longitudinal Position.
DCM_TAG_TableTopLateralPosition	Table Top Lateral Position.
DCM_TAG_IsocenterPosition	Isocenter Position.
DCM_TAG_SurfaceEntryPoint	Surface Entry Point.
DCM_TAG_SourceToSurfaceDistance	Source to Surface Distance.
DCM_TAG_CumulativeMetersetWeight	Cumulative Meterset Weight.
DCM_TAG_TableTopPitchAngle	Table Top Pitch Angle.
DCM_TAG_TableTopPitchRotationDirection	Table Top Pitch Rotation Direction.
DCM_TAG_TableTopRollAngle	Table Top Roll Angle.
DCM_TAG_TableTopRollRotationDirection	Table Top Roll Rotation Direction.
DCM_TAG_HeadFixationAngle	Head Fixation Angle.
DCM_TAG_GantryPitchAngle	Gantry Pitch Angle.
DCM_TAG_GantryPitchRotationDirection	Gantry Pitch Rotation Direction.
DCM_TAG_GantryPitchAngleTolerance	Gantry Pitch Angle Tolerance.
DCM_TAG_PatientSetupSequence	Patient Setup Sequence.
DCM_TAG_PatientSetupNumber	Patient Setup Number.
DCM_TAG_PatientSetupLabel	Patient Setup Label.
DCM_TAG_PatientAdditionalPosition	Patient Additional Position.
DCM_TAG_FixationDeviceSequence	Fixation Device Sequence.
DCM_TAG_FixationDeviceType	Fixation Device Type.
DCM_TAG_FixationDeviceLabel	Fixation Device Label.
DCM_TAG_FixationDeviceDescription	Fixation Device Description.
DCM_TAG_FixationDevicePosition	Fixation Device Position.
DCM_TAG_FixationDevicePitchAngle	Fixation Device Pitch Angle.
DCM_TAG_FixationDeviceRollAngle	Fixation Device Roll Angle.
DCM_TAG_ShieldingDeviceSequence	Shielding Device Sequence.
DCM_TAG_ShieldingDeviceType	Shielding Device Type.
DCM_TAG_ShieldingDeviceLabel	Shielding Device Label.
DCM_TAG_ShieldingDeviceDescription	Shielding Device Description.
DCM_TAG_ShieldingDevicePosition	Shielding Device Position.
DCM_TAG_SetupTechnique	Setup Technique.
DCM_TAG_SetupTechniqueDescription	Setup Technique Description.
DCM_TAG_SetupDeviceSequence	Setup Device Sequence.
DCM_TAG_SetupDeviceType	Setup Device Type.
DCM_TAG_SetupDeviceLabel	Setup Device Label.
DCM_TAG_SetupDeviceDescription	Setup Device Description.
DCM_TAG_SetupDeviceParameter	Setup Device Parameter.
DCM_TAG_SetupReferenceDescription	Setup Reference Description.
DCM_TAG_TableTopVerticalSetupDisplacement	Table Top Vertical Setup Displacement.

DCM_TAG_TableTopLongitudinalSetupDisplacement	Table Top Longitudinal Setup Displacement.
DCM_TAG_TableTopLogitudinalSetupDisplacement	Table Top Longitudinal Setup Displacement. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_TableTopLateralSetupDisplacement	Table Top Lateral Setup Displacement.
DCM_TAG_BrachyTreatmentTechnique	Brachy Treatment Technique.
DCM_TAG_BrachyTreatmentType	Brachy Treatment Type.
DCM_TAG_TreatmentMachineSequence	Treatment Machine Sequence.
DCM_TAG_SourceSequence	Source Sequence.
DCM_TAG_SourceNumber	Source Number.
DCM_TAG_SourceType	Source Type.
DCM_TAG_SourceManufacturer	Source Manufacturer.
DCM_TAG_ActiveSourceDiameter	Active Source Diameter.
DCM_TAG_ActiveSourceLength	Active Source Length.
DCM_TAG_SourceEncapsulationNominalThickness	Source Encapsulation Nominal Thickness.
DCM_TAG_SourceEncapsulationNominalTransmission	Source Encapsulation Nominal Transmission.
DCM_TAG_SourceIsotopeName	Source Isotope Name.
DCM_TAG_SourceIsotopeHalfLife	Source Isotope Half Life.
DCM_TAG_SourceStrengthUnits	Source Strength Units.
DCM_TAG_ReferenceAirKermaRate	Reference Air Kerma Rate.
DCM_TAG_SourceStrength	Source Strength.
DCM_TAG_SourceStrengthReferenceDate	Source Strength Reference Date.
DCM_TAG_SourceStrengthAirKermaRateReferenceDateDA1	Source Strength Reference Date. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SourceStrengthReferenceTime	Source Strength Reference Time.
DCM_TAG_SourceStrengthAirKermaRateReferenceTimeTM1	Source Strength Reference Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ApplicationSetupSequence	Application Setup Sequence.
DCM_TAG_ApplicationSetupType	Application Setup Type.
DCM_TAG_ApplicationSetupNumber	Application Setup Number.
DCM_TAG_ApplicationSetupName	Application Setup Name.
DCM_TAG_ApplicationSetupManufacturer	Application Setup Manufacturer.
DCM_TAG_TemplateNumber	Template Number.
DCM_TAG_TemplateType	Template Type.
DCM_TAG_TemplateName	Template Name.
DCM_TAG_TotalReferenceAirKerma	Total Reference Air Kerma.

DCM_TAG_BrachyAccessoryDeviceSequence	Brachy Accessory Device Sequence.
DCM_TAG_BrachyAccDeviceSequence	Brachy Accessory Device Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BrachyAccessoryDeviceNumber	Brachy Accessory Device Number.
DCM_TAG_BrachyAccDeviceNumber	Brachy Accessory Device Number. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BrachyAccessoryDeviceID	Brachy Accessory Device ID.
DCM_TAG_BrachyAccDeviceID	Brachy Accessory Device ID. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BrachyAccessoryDeviceType	Brachy Accessory Device Type.
DCM_TAG_BrachyAccDeviceType	Brachy Accessory Device Type. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BrachyAccessoryDeviceName	Brachy Accessory Device Name.
DCM_TAG_BrachyAccDeviceName	Brachy Accessory Device Name. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BrachyAccessoryDeviceNominalThickness	Brachy Accessory Device Nominal Thickness.
DCM_TAG_BrachyAccDeviceNominalThickness	Brachy Accessory Device Nominal Thickness. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_BrachyAccessoryDeviceNominalTransmission	Brachy Accessory Device Nominal Transmission.
DCM_TAG_BrachyAccDeviceNominalTransmission	Brachy Accessory Device Nominal Transmission. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ChannelSequence	Channel Sequence.
DCM_TAG_BrachyChannelSequence	Channel Sequence. This tag

	name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ChannelNumber	Channel Number.
DCM_TAG_BrachyChannelNumber	Channel Number. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ChannelLength	Channel Length.
DCM_TAG_BrachyChannelLength	Channel Length. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ChannelTotalTime	Channel Total Time.
DCM_TAG_BrachyChannelTotalTime	Channel Total Time. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_SourceMovementType	Source Movement Type.
DCM_TAG_NumberOfPulses	Number of Pulses.
DCM_TAG_PulseRepetitionInterval	Pulse Repetition Interval.
DCM_TAG_SourceApplicatorNumber	Source Applicator Number.
DCM_TAG_SourceApplicatorID	Source Applicator ID.
DCM_TAG_SourceApplicatorType	Source Applicator Type.
DCM_TAG_SourceApplicatorName	Source Applicator Name.
DCM_TAG_SourceApplicatorLength	Source Applicator Length.
DCM_TAG_SourceApplicatorManufacturer	Source Applicator Manufacturer.
DCM_TAG_SourceApplicatorWallNominalThickness	Source Applicator Wall Nominal Thickness.
DCM_TAG_SourceApplicatorWallNominalTransmission	Source Applicator Wall Nominal Transmission.
DCM_TAG_SourceApplicatorStepSize	Source Applicator Step Size.
DCM_TAG_TransferTubeNumber	Transfer Tube Number.
DCM_TAG_TransferTubeLength	Transfer Tube Length.
DCM_TAG_ChannelShieldSequence	Channel Shield Sequence.
DCM_TAG_ChannelShieldNumber	Channel Shield Number.
DCM_TAG_ChannelShieldID	Channel Shield ID.
DCM_TAG_ChannelShieldName	Channel Shield Name.
DCM_TAG_ChannelShieldNominalThickness	Channel Shield Nominal Thickness.
DCM_TAG_ChannelShieldNominalTransmission	Channel Shield Nominal Transmission.
DCM_TAG_FinalCumulativeTimeWeight	Final Cumulative Time Weight.
DCM_TAG_BrachyControlPointSequence	Brachy Control Point Sequence.
DCM_TAG_ControlPointRelativePosition	Control Point Relative Position.
DCM_TAG_ControlPoint3dPosition	Control Point 3D Position.
DCM_TAG_CumulativeTimeWeight	Cumulative Time Weight.

DCM_TAG_CompensatorDivergence	Compensator Divergence.
DCM_TAG_CompensatorMountingPosition	Compensator Mounting Position.
DCM_TAG_SourceToCompensatorDistance	Source to Compensator Distance.
DCM_TAG_TotalCompensatorTrayWaterEquivalentThickness	Total Compensator Tray Water-Equivalent Thickness.
DCM_TAG_IsocentertoCompensatorTrayDistance	Isocenter to Compensator Tray Distance.
DCM_TAG_CompensatorColumnOffset	Compensator Column Offset.
DCM_TAG_IsocentertoCompensatorDistances	Isocenter to Compensator Distances.
DCM_TAG_CompensatorRelativeStoppingPowerRatio	Compensator Relative Stopping Power Ratio.
DCM_TAG_CompensatorMillingToolDiameter	Compensator Milling Tool Diameter.
DCM_TAG_IonRangeCompensatorSequence	Ion Range Compensator Sequence.
DCM_TAG_CompensatorDescription	Compensator Description.
DCM_TAG_RadiationMassNumber	Radiation Mass Number.
DCM_TAG_RadiationAtomicNumber	Radiation Atomic Number.
DCM_TAG_RadiationChargeState	Radiation Charge State.
DCM_TAG_ScanMode	Scan Mode.
DCM_TAG_VirtualSourceAxisDistances	Virtual Source-Axis Distances.
DCM_TAG_SnoutSequence	Snout Sequence.
DCM_TAG_SnoutPosition	Snout Position.
DCM_TAG_SnoutID	Snout Identifier.
DCM_TAG_NumberofRangeShifters	Number of Range Shifters.
DCM_TAG_RangeShifterSequence	Range Shifter Sequence.
DCM_TAG_RangeShifterNumber	Range Shifter Number.
DCM_TAG_RangeShifterID	Range Shifter ID.
DCM_TAG_RangeShifterType	Range Shifter Type.
DCM_TAG_RangeShifterDescription	Range Shifter Description.
DCM_TAG_NumberofLateralSpreadingDevices	Number of Lateral Spreading Devices.
DCM_TAG_LateralSpreadingDeviceSequence	Lateral Spreading Device Sequence.
DCM_TAG_LateralSpreadingDeviceNumber	Lateral Spreading Device Number.
DCM_TAG_LateralSpreadingDeviceID	Lateral Spreading Device ID.
DCM_TAG_LateralSpreadingDeviceType	Lateral Spreading Device Type.
DCM_TAG_LateralSpreadingDeviceDescription	Lateral Spreading Device Description.
DCM_TAG_LateralSpreadingDeviceWaterEquivalentThickness	Lateral Spreading Device Water Equivalent Thickness.
DCM_TAG_NumberofRangeModulators	Number of Range Modulators.
DCM_TAG_RangeModulatorSequence	Range Modulator Sequence.
DCM_TAG_RangeModulatorNumber	Range Modulator Number.
DCM_TAG_RangeModulatorID	Range Modulator ID.
DCM_TAG_RangeModulatorType	Range Modulator Type.
DCM_TAG_RangeModulatorDescription	Range Modulator Description.
DCM_TAG_BeamCurrentModulationID	Beam Current Modulation ID.
DCM_TAG_PatientSupportType	Patient Support Type.

DCM_TAG_PatientSupportID	Patient Support ID.
DCM_TAG_PatientSupportAccessoryCode	Patient Support Accessory Code.
DCM_TAG_FixationLightAzimuthalAngle	Fixation Light Azimuthal Angle.
DCM_TAG_FixationLightPolarAngle	Fixation Light Polar Angle.
DCM_TAG_MetersetRate	Meterset Rate.
DCM_TAG_RangeShifterSettingsSequence	Range Shifter Settings Sequence.
DCM_TAG_RangeShifterSetting	Range Shifter Setting.
DCM_TAG_IsocentertoRangeShifterDistance	Isocenter to Range Shifter Distance.
DCM_TAG_RangeShifterWaterEquivalentThickness	Range Shifter Water Equivalent Thickness.
DCM_TAG_LateralSpreadingDeviceSettingsSequence	Lateral Spreading Device Settings Sequence.
DCM_TAG_LateralSpreadingDeviceSetting	Lateral Spreading Device Setting.
DCM_TAG_IsocentertoLateralSpreadingDeviceDistance	Isocenter to Lateral Spreading Device Distance.
DCM_TAG_RangeModulatorSettingsSequence	Range Modulator Settings Sequence.
DCM_TAG_RangeModulatorGatingStartValue	Range Modulator Gating Start Value.
DCM_TAG_RangeModulatorGatingStopValue	Range Modulator Gating Stop Value.
DCM_TAG_RangeModulatorGatingStartWaterEquivalentThickness	Range Modulator Gating Start Water Equivalent Thickness.
DCM_TAG_RangeModulatorGatingStartWaterEquivalent	Range Modulator Gating Start Water Equivalent Thickness. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_RangeModulatorGatingStopWaterEquivalentThickness	Range Modulator Gating Stop Water Equivalent Thickness.
DCM_TAG_RangeModulatorGatingStopWaterEquivalent	Range Modulator Gating Stop Water Equivalent Thickness. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_IsocentertoRangeModulatorDistance	Isocenter to Range Modulator Distance.
DCM_TAG_ScanSpotTuneID	Scan Spot Tune ID.
DCM_TAG_NumberofScanSpotPositions	Number of Scan Spot Positions.
DCM_TAG_ScanSpotPositionMap	Scan Spot Position Map.
DCM_TAG_ScanSpotMetersetWeights	Scan Spot Meterset Weights.
DCM_TAG_ScanningSpotSize	Scanning Spot Size.
DCM_TAG_NumberofPaintings	Number of Paintings.
DCM_TAG_IonToleranceTableSequence	Ion Tolerance Table Sequence.
DCM_TAG_IonBeamSequence	Ion Beam Sequence.
DCM_TAG_IonBeamLimitingDeviceSequence	Ion Beam Limiting Device Sequence.
DCM_TAG_IonBlockSequence	Ion Block Sequence.

DCM_TAG_IonControlPointSequence	Ion Control Point Sequence.
DCM_TAG_IonWedgeSequence	Ion Wedge Sequence.
DCM_TAG_IonWedgePositionSequence	Ion Wedge Position Sequence.
DCM_TAG_ReferencedSetupImageSequence	Referenced Setup Image Sequence.
DCM_TAG_SetupImageComment	Setup Image Comment.
DCM_TAG_MotionSynchronizationSequence	Motion Synchronization Sequence.
DCM_TAG_ControlPointOrientation	Control Point Orientation.
DCM_TAG_GeneralAccessorySequence	General Accessory Sequence.
DCM_TAG_GeneralAccessoryID	General Accessory ID.
DCM_TAG_GeneralAccessoryDescription	General Accessory Description.
DCM_TAG_GeneralAccessoryType	General Accessory Type.
DCM_TAG_GeneralAccessoryNumber	General Accessory Number.
DCM_TAG_Group300CLength	Group 300C Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedRtPlanSequence	Referenced RT Plan Sequence.
DCM_TAG_ReferencedBeamSequence	Referenced Beam Sequence.
DCM_TAG_ReferencedBeamNumber	Referenced Beam Number.
DCM_TAG_ReferencedReferenceImageNumber	Referenced Reference Image Number.
DCM_TAG_StartCumulativeMetersetWeight	Start Cumulative Meterset Weight.
DCM_TAG_EndCumulativeMetersetWeight	End Cumulative Meterset Weight.
DCM_TAG_ReferencedBrachyApplicationSetupSequence	Referenced Brachy Application Setup Sequence.
DCM_TAG_ReferencedBrachyAppSetupSequence	Referenced Brachy Application Setup Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedBrachyApplicationSetupNumber	Referenced Brachy Application Setup Number.
DCM_TAG_ReferencedBrachyAppSetupNumber	Referenced Brachy Application Setup Number. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedSourceNumber	Referenced Source Number.
DCM_TAG_ReferencedFractionGroupSequence	Referenced Fraction Group Sequence.
DCM_TAG_ReferencedFractionGroupNumber	Referenced Fraction Group Number.
DCM_TAG_ReferencedVerificationImageSequence	Referenced Verification Image Sequence.
DCM_TAG_ReferencedReferenceImageSequence	Referenced Reference Image Sequence.
DCM_TAG_ReferencedDoseReferenceSequence	Referenced Dose Reference Sequence.
DCM_TAG_ReferencedDoseReferenceNumber	Referenced Dose Reference

	Number.
DCM_TAG_BrachyReferencedDoseReferenceSequence	Brachy Referenced Dose Reference Sequence.
DCM_TAG_ReferencedStructureSetSequence	Referenced Structure Set Sequence.
DCM_TAG_ReferencedPatientSetupNumber	Referenced Patient Setup Number.
DCM_TAG_ReferencedDoseSequence	Referenced Dose Sequence.
DCM_TAG_ReferencedToleranceTableNumber	Referenced Tolerance Table Number.
DCM_TAG_ReferencedBolusSequence	Referenced Bolus Sequence.
DCM_TAG_ReferencedWedgeNumber	Referenced Wedge Number.
DCM_TAG_ReferencedCompensatorNumber	Referenced Compensator Number.
DCM_TAG_ReferencedBlockNumber	Referenced Block Number.
DCM_TAG_ReferencedControlPointIndex	Referenced Control Point Index.
DCM_TAG_ReferencedControlPoint	Referenced Control Point Index. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ReferencedControlPointSequence	Referenced Control Point Sequence.
DCM_TAG_ReferencedStartControlPointIndex	Referenced Start Control Point Index.
DCM_TAG_ReferencedStopControlPointIndex	Referenced Stop Control Point Index.
DCM_TAG_ReferencedRangeShifterNumber	Referenced Range Shifter Number.
DCM_TAG_ReferencedLateralSpreadingDeviceNumber	Referenced Lateral Spreading Device Number.
DCM_TAG_ReferencedRangeModulatorNumber	Referenced Range Modulator Number.
DCM_TAG_Group300ELength	Group 300E Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ApprovalStatus	Approval Status.
DCM_TAG_ReviewDate	Review Date.
DCM_TAG_ReviewTime	Review Time.
DCM_TAG_ReviewerName	Reviewer Name.
DCM_TAG_Group4000Length	Group 4000 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Arbitrary	Arbitrary. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TextComments	Text Comments. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group4008Length	Group 4008 Length. This tag is marked as retired in DICOM

	specification. See DICOM specification for alternatives.
DCM_TAG_ResultsID	Results ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ResultsIDIssuer	Results ID Issuer. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencedInterpretationSequence	Referenced Interpretation Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationRecordedDate	Interpretation Recorded Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationRecordedTime	Interpretation Recorded Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationRecorder	Interpretation Recorder. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ReferencetoRecordedSound	Reference to Recorded Sound. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationTranscriptionDate	Interpretation Transcription Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationTranscriptionTime	Interpretation Transcription Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationTranscriber	Interpretation Transcriber. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationText	Interpretation Text. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationAuthor	Interpretation Author. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationApproverSequence	Interpretation Approver Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationApprovalDate	Interpretation Approval Date. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.

DCM_TAG_InterpretationApprovalTime	Interpretation Approval Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PhysicianApprovingInterpretation	Physician Approving Interpretation. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationDiagnosisDescription	Interpretation Diagnosis Description. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationDiagnosisCodeSequence	Interpretation Diagnosis Code Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ResultsDistributionListSequence	Results Distribution List Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DistributionName	Distribution Name. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DistributionAddress	Distribution Address. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationID	Interpretation ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationIDIssuer	Interpretation ID Issuer. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationTypeID	Interpretation Type ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_InterpretationStatusID	Interpretation Status ID. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Impressions	Impressions. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_ResultsComments	Results Comments. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group4FFELength	Group 4FFE Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MacParametersSequence	MAC Parameters Sequence.
DCM_TAG_Group50xxLength	Group 50xx Length. This tag is

	marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveDimensions	Curve Dimensions. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfPoints	Number of Points. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TypeOfData	Type of Data. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveDescription	Curve Description. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AxisUnits	Axis Units. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AxisLabels	Axis Labels. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DataValueRepresentation	Data Value Representation. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MinimumCoordinateValue	Minimum Coordinate Value. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_MaximumCoordinateValue	Maximum Coordinate Value. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveRange	Curve Range. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveDataDescriptor	Curve Data Descriptor. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CoordinateStartValue	Coordinate Start Value. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CoordinateStepValue	Coordinate Step Value. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveActivationLayer	Curve Activation Layer. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AudioType	Audio Type. This tag is marked as retired in DICOM specification. See DICOM specification for

	alternatives.
DCM_TAG_AudioSampleFormat	Audio Sample Format. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfChannels	Number of Channels. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfSamples	Number of Samples. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SampleRate	Sample Rate. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_TotalTime	Total Time. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AudioSampleData	Audio Sample Data. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_AudioComments	Audio Comments. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveLabel	Curve Label. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveReferencedOverlaySequence	Curve Referenced Overlay Sequence. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveReferencedOverlayGroup	Curve Referenced Overlay Group. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CurveData	Curve Data. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group5200Length	Group 5200 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_SharedFunctionalGroupsSequence	Shared Functional Groups Sequence.
DCM_TAG_SharedFunctionalGroupSequence	Shared Functional Groups Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_PerFrameFunctionalGroupsSequence	Per-frame Functional Groups Sequence.

DCM_TAG_PerFrameFunctionalGroupSequence	Per-frame Functional Groups Sequence. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_Group5400Length	Group 5400 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_WaveformSequence	Waveform Sequence.
DCM_TAG_ChannelMinimumValue	Channel Minimum Value.
DCM_TAG_ChannelMaximumValue	Channel Maximum Value.
DCM_TAG_WaveformBitsAllocated	Waveform Bits Allocated.
DCM_TAG_WaveformSampleInterpretation	Waveform Sample Interpretation.
DCM_TAG_WaveformPaddingValue	Waveform Padding Value.
DCM_TAG_WaveformData	Waveform Data.
DCM_TAG_Group5600Length	Group 5600 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_FirstOrderPhaseCorrectionAngle	First Order Phase Correction Angle.
DCM_TAG_SpectroscopyData	Spectroscopy Data.
DCM_TAG_Group60xxLength	Group 60xx Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayRows	Overlay Rows.
DCM_TAG_OverlayColumns	Overlay Columns.
DCM_TAG_OverlayPlanes	Overlay Planes. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_NumberOfFramesInOverlay	Number of Frames in Overlay.
DCM_TAG_OverlayDescription	Overlay Description.
DCM_TAG_OverlayType	Overlay Type.
DCM_TAG_OverlaySubtype	Overlay Subtype.
DCM_TAG_OverlayOrigin	Overlay Origin.
DCM_TAG_Origin	Overlay Origin. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ImageFrameOrigin	Image Frame Origin.
DCM_TAG_OverlayPlaneOrigin	Overlay Plane Origin. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayCompressionCode	Overlay Compression Code. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayCompressionOriginator	Overlay Compression Originator. This tag is marked as retired in

	DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayCompressionLabel	Overlay Compression Label. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayCompressionDescription	Overlay Compression Description. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayCompressionStepPointers	Overlay Compression Step Pointers. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayRepeatInterval	Overlay Repeat Interval. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayBitsGrouped	Overlay Bits Grouped. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayBitsAllocated	Overlay Bits Allocated.
DCM_TAG_BitAllocated	Overlay Bits Allocated. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OverlayBitPosition	Overlay Bit Position.
DCM_TAG_BitPosition	Overlay Bit Position. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_OverlayFormat	Overlay Format. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayLocation	Overlay Location. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayCodeLabel	Overlay Code Label. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayNumberOfTables	Overlay Number of Tables. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayCodeTableLocation	Overlay Number of Tables. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayBitsForCodeWord	Overlay Bits For Code Word. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.

DCM_TAG_OverlayActivationLayer	Overlay Activation Layer.
DCM_TAG_OverlayDescriptorGray	Overlay Descriptor - Gray. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayDescriptorRed	Overlay Descriptor - Red. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayDescriptorGreen	Overlay Descriptor - Green. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlayDescriptorBlue	Overlay Descriptor - Blue. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlaysGray	Overlays - Gray. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlaysRed	Overlays - Red. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlaysGreen	Overlays - Green. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_OverlaysBlue	Overlays - Blue. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_RoiArea	The ROI Area.
DCM_TAG_RoiMean	The ROI Mean.
DCM_TAG_RoiStandardDeviation	ROI Standard Deviation.
DCM_TAG_OverlayLabel	Overlay Label.
DCM_TAG_OverlayData	Overlay Data.
DCM_TAG_OverlayComments	Overlay Comments. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group7FE0Length	Group 7FE0 Length. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_PixelData	Pixel Data.
DCM_TAG_CoefficientsSDVN	Coefficients SDVN. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CoefficientsSDHN	Coefficients SDHN. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_CoefficientsSDDN	Coefficients SDDN. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_Group7FFFLength	Group 7FFF Length. This tag is

	marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_VariablePixelData	Variable Pixel Data. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_VariableNextDataGroup	Variable Next Data Group. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_VariableCoefficientsSDVN	Variable Coefficients SDVN. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_VariableCoefficientsSDHN	Variable Coefficients SDHN. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_VariableCoefficientsSDDN	Variable Coefficients SDDN. This tag is marked as retired in DICOM specification. See DICOM specification for alternatives.
DCM_TAG_DigitalSignaturesSequence	Digital Signatures Sequence.
DCM_TAG_DataSetTrailingPadding	Data Set Trailing Padding.
DCM_TAG_DatasetPadding	Data Set Trailing Padding. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_Item	Item marker.
DCM_TAG_ItemItem	Item marker. This tag name has been deprecated and will be removed from the public API in a future release. Please use the tag with the same value defined in the previous line.
DCM_TAG_ItemDelimitationItem	Item Delimitation Item.
DCM_TAG_SequenceDelimitationItem	Sequence Delimitation Item.
DCM_TAG_DataStreamEncodingFragment	Data Stream Encoding Fragment.

1.3.2.4.9 enumIGMedTS

Specifies DICOM Transfer Syntaxes.

Values:

MED_DCM_TS_IMPLICIT_VR_LE	Implicit VR Little Endian: Default Transfer Syntax for DICOM (uncompressed).
MED_DCM_TS_EXPLICIT_VR_LE	Explicit VR Little Endian (uncompressed).
MED_DCM_TS_EXPLICIT_VR_BE	Explicit VR Big Endian (uncompressed).
MED_DCM_TS_DEFLATED_EXPLICIT_VR_LE	Deflated Explicit VR Little Endian.
MED_DCM_TS_JPEG_BASELINE_PR_1	JPEG Baseline (Process 1): Default Transfer Syntax for Lossy JPEG 8 Bit Image Compression.
MED_DCM_TS_JPEG_EXTENDED_PR_2_4	JPEG Extended (Process 2 & 4): Default Transfer Syntax for Lossy JPEG 12 Bit Image Compression (Process 4 only).
MED_DCM_TS_JPEG_EXTENDED_PR_3_5	JPEG Extended (Process 3 & 5) (Retired).
MED_DCM_TS_JPEG_SPECTRAL_NONH_PR_6_8	JPEG Spectral Selection, Non-Hierarchical (Process 6 & 8) (Retired).
MED_DCM_TS_JPEG_SPECTRAL_NONH_PR_7_9	JPEG Spectral Selection, Non-Hierarchical (Process 7 & 9) (Retired).
MED_DCM_TS_JPEG_FULL_PROG_NONH_PR_10_12	JPEG Full Progression, Non-Hierarchical (Process 10 & 12) (Retired).
MED_DCM_TS_JPEG_FULL_PROG_NONH_PR_11_13	JPEG Full Progression, Non-Hierarchical (Process 11 & 13) (Retired).
MED_DCM_TS_JPEG_LOSSLESS_NONH_PR_14	JPEG Lossless, Non-Hierarchical (Process 14).
MED_DCM_TS_JPEG_LOSSLESS_NONH_PR_15	JPEG Lossless, Non-Hierarchical (Process 15) (Retired).
MED_DCM_TS_JPEG_EXTENDED_HIER_PR_16_18	JPEG Extended, Hierarchical (Process 16 & 18) (Retired).
MED_DCM_TS_JPEG_EXTENDED_HIER_PR_17_19	JPEG Extended, Hierarchical (Process 17 & 19) (Retired).
MED_DCM_TS_JPEG_SPECTRAL_HIER_PR_20_22	JPEG Spectral Selection, Hierarchical (Process 20 & 22) (Retired).
MED_DCM_TS_JPEG_SPECTRAL_HIER_PR_21_23	JPEG Spectral Selection, Hierarchical (Process 21 & 23) (Retired).
MED_DCM_TS_JPEG_FULL_PROG_HIER_PR_24_26	JPEG Full Progression, Hierarchical (Process 24 & 26) (Retired).
MED_DCM_TS_JPEG_FULL_PROG_HIER_PR_25_27	JPEG Full Progression, Hierarchical (Process 25 & 27) (Retired).
MED_DCM_TS_JPEG_LOSSLESS_HIER_PR_28	JPEG Lossless, Hierarchical (Process 28) (Retired).
MED_DCM_TS_JPEG_LOSSLESS_HIER_PR_29	JPEG Lossless, Hierarchical (Process 29) (Retired).
MED_DCM_TS_JPEG_LOSSLESS_NONH_FIRSTORDER_PR_14	JPEG Lossless, Non-Hierarchical, First-Order Prediction (Process 14 [Selection Value 1]): Default Transfer Syntax for Lossless JPEG Image Compression.
MED_DCM_TS_JPEG_LS_LOSSLESS	JPEG-LS Lossless Image Compression.
MED_DCM_TS_JPEG_LS_LOSSY	JPEG-LS Lossy (Near-Lossless) Image Compression.
MED_DCM_TS_JPEG_2K_LOSSLESS_ONLY	JPEG 2000 Image Compression (Lossless Only).
MED_DCM_TS_JPEG_2K	JPEG 2000 Image Compression.
MED_DCM_TS_JPEG_2000_PART_2_MULTI_COMPONENT_IMAGE_COMPRESSION_Lossless_Only	JPEG 2000 Part 2 Multi-component Image Compression (Lossless Only).
MED_DCM_TS_JPEG_2000_PART_2_MULTI_COMPONENT_IMAGE_COMPRESSION	JPEG 2000 Part 2 Multi-component Image Compression.
MED_DCM_TS_JPIP_REFERENCED	JPIP Referenced.
MED_DCM_TS_JPIP_REFERENCED_DEFLATE	JPIP Referenced Deflate.
MED_DCM_TS_MPEG2_MAIN_PROFILE	MPEG2 Main Profile @ Main Level.
MED_DCM_TS_MPEG2_MAIN_PROFILE_HIGH_LEVEL	MPEG2 Main Profile @ High Level.
MED_DCM_TS_RLE	RLE Lossless.
MED_DCM_TS_RFC_2557_MIME_ENCAPSULATION	RFC 2557 MIME encapsulation.
MED_DCM_TS_XML	XML Encoding.
MED_DCM_TS_DEFAULT	DICOM default transfer syntax: Implicit VR Little Endian.
MED_DCM_TS_UNKNOWN	Unknown transfer syntax.
MED_DCM_TS_NULL	Unknown transfer syntax.
MED_DCM_TS_PART_10	Autodetect. This value is used with DICOM LoadSyntax format option, to allow detection of DICOM Part10 compliant images.
MED_DCM_TS_AUTODETECT	Autodetect. This value is used with DICOM LoadSyntax format option, to allow automatic detection of transfer syntax.
MED_DCM_TS_JPEG_LOSSY	JPEG Baseline (Process 1): Default Transfer Syntax for Lossy JPEG 8 Bit Image Compression.
MED_DCM_TS_JPEG_LOSSLESS	JPEG Lossless, Non-Hierarchical (Process 14).
MED_DCM_TS_JPEG_LOSSLESS_FIRSTORDER	Alias for MED_DCM_TS_JPEG_LOSSLESS_NONH_FIRSTORDER_PR_14.

MED_DCM_TS_JPEG_BASELINE_PR_1_ONLY	Only used for image saving. Specifies JPEG baseline (process 1) Transfer Syntax.

1.3.2.4.10 enumIGMedVR

Identifies DICOM Value Representations.

Values:

MED_DCM_VR_AE	Application Entity.
MED_DCM_VR_AS	Age String.
MED_DCM_VR_AT	Attribute Tag.
MED_DCM_VR_CS	Code String.
MED_DCM_VR_DA	Date.
MED_DCM_VR_DS	Decimal String.
MED_DCM_VR_DT	Date Time.
MED_DCM_VR_FL	Floating Point Single.
MED_DCM_VR_FD	Floating Point Double.
MED_DCM_VR_IS	Integer String.
MED_DCM_VR_LO	Long String.
MED_DCM_VR_LT	Long Text.
MED_DCM_VR_OB	Other Byte String.
MED_DCM_VR_OF	Other Float String.
MED_DCM_VR_OW	Other Word String.
MED_DCM_VR_PN	Person Name.
MED_DCM_VR_SH	Short String.
MED_DCM_VR_SL	Signed Long.
MED_DCM_VR_SQ	Sequence of Items.
MED_DCM_VR_SS	Signed Short.
MED_DCM_VR_ST	Short Text.
MED_DCM_VR_TM	Time.
MED_DCM_VR_UI	Unique Identifier.
MED_DCM_VR_UL	Unsigned Long.
MED_DCM_VR_US	Unsigned Short.
MED_DCM_VR_UN	Unknown.
MED_DCM_VR_UT	Unlimited text.
MED_DCM_VR_NONE	VR is not known.

1.3.2.4.11 enumIGMedVRRestriction

Identifies Value Representation length restrictions.

Values:

MED_DCM_LEN_MAX	Up to this maximum.
MED_DCM_LEN_FIXED	Fixed Length.
MED_DCM_LEN_TS	Depends on Transfer Syntax.
MED_DCM_LEN_NA	Not applicable.
MED_DCM_LEN_UNLIMITED	No max length.

1.3.3 PDF Component API Reference

The ImageGear PDF component is responsible for the PDF functionality. The ImageGear PDF component exposes handles and objects described in the following table. The general PDF layer is implemented via "PDF" objects (for example, HIG_PDF_DOC). The PDF editing layer is implemented via "PDE" objects (for example, HIG_PDE_CONTENT).

This chapter references the Adobe PDF 1.7 specification, which can be downloaded from here: http://www.adobe.com/devnet/pdf/pdf_reference.html

The ImageGear PDF Component API reference is grouped as follows:

PDF General Functions Reference	PDF functions that provide general ImageGear PDF Component functionality.
PDF Callback Functions Reference	ImageGear PDF Component callback functions.
PDF Macro Reference	ImageGear PDF Component macros.
PDF Objects Reference	ImageGear PDF Component objects.
PDF Structures Reference	ImageGear PDF Component structures.
PDF Enumerations Reference	ImageGear PDF Component enumerations.

1.3.3.1 PDF Component Functions Reference

This section provides information about the General group of functions.

- IG_PDE_get_default_gstate
- IG_PDF_get_host_encoding
- IG_PDF_initialize
- IG_PDF_register_authproc
- IG_PDF_terminate
- IG_PDF_text_extract
- IG_PDF_translate_to_host
- IG_PDF_translate_to_pdf

1.3.3.1.1 IG_PDE_get_default_gstate

Fills out a LPAT_PDE_GRAPHICSTATE structure with the default graphic state.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_get_default_gstate(
        LPAT_PDE_GRAPHICSTATE lpGstate
);
```

Arguments:

Name	Type	Description
lpGstate	LPAT_PDE_GRAPHICSTATE	Pointer to AT_PDE_GRAPHICSTATE structure with the default graphic state.

Return Value:

Error count

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Non-NULL objects in the graphic state, such as the fill and stroke color spaces, have their reference counts incremented by this function. Be sure to release these non-NULL objects when disposing of lpGstate.

1.3.3.1.2 IG_PDF_get_host_encoding

Indicates what kind of host encoding a system uses: Roman or non-Roman.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_get_host_encoding(
        LPVOID* lpHostEncoding
);
```

Arguments:

Name	Type	Description
lpHostEncoding	LPVOID*	Returns 0 for a Roman system; nonzero for a non-Roman system (a structure that depends on the host encoding). Users should simply test whether this value is 0 or not.

Return Value:

Error count

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Non-Roman is also known as CJK-capable, that is, capable of handling multi-byte character sets, such as Chinese, Japanese, or Korean.

Host encoding is a platform-dependent encoding for the host machine. For non-UNIX Roman systems, it is MacRomanEncoding in Mac OS and WinAnsiEncoding in Windows. In UNIX (except HP-UX) Roman systems, it is ISO8859-1 (ISO Latin-1); for HP-UX, it is HP-ROMAN8. See Appendix D in the PDF Reference for descriptions of MacRomanEncoding, WinAnsiEncoding, and PDFDocEncoding.

For non-Roman systems, the host encoding may be a variety of encodings, which are defined by a CMap (character map). See Section 5.6.4 in the PDF Reference for a list of predefined CMaps.

1.3.3.1.3 IG_PDF_initialize

This function is used to initialize the ImageGear PDF Component.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_initialize(
        LPVOID lpVoid
);
```

Arguments:

Name	Type	Description
lpVoid	LPVOID	Reserved, must be set to NULL.

Return Value:

Error count

Supported Raster Image Formats:

This function does not process image pixels.

Example:

Remarks:

To initialize the ImageGear PDF component it needs to be attached to the core ImageGear and then IG_PDF_initialize() function must be called.

> For a multi-threaded application, you must call IG_PDF_initialize and IG_PDF_terminate in the main thread and in each worker thread which uses PDF component. See Single- and Multi-Threaded Applications for more information.

The following resource content is required by the ImageGear PDF component initialization routine:

Resource\PDF\CIDFont\	PDF CID fonts directory.
Resource\PDF\CMap\	PDF font CMaps directory.
Resource\PDF\Font\	PDF fonts directory.
Resource\PDF\Unicode\	PDF unicode mappings directory.
Resource\PS\ColorRendering\	Color rendering PostScript.
Resource\PS\ICCProfiles\	Directory containing the ICC profiles that allow using the Adobe® Color Engine® (ACE®). The profiles in this directory must be placed in the system folder named, which on Windows is named \Windows\System32\Color.
Resource\PS\Fonts\	PS fonts directory.
Resource\PS\ProcSet\	PostScript procedures.
Resource\PS\ps.vm	A file for initializing the PostScript Interpreter's virtual memory.
Resource\PS\startupNORM.ps	Startup PostScript program used to initialize the PostScript Interpreter.
Resource\PS\superatm.db	Adobe® Type Manager® (ATM®) database used to substitute missing fonts.

ImageGear PDF component uses the following PDF global control parameters to locate resource content:

PDF.PDF_RESOURCE_PATH	Path to the Resource\PDF directory
PDF.PS_RESOURCE_PATH	Path to the Resource\PS directory
PDF.HOST_FONT_PATH	Path to the system font directory

PDF.TMP_PATH	Path to the TEMP directory

Examples provided below demonstrate setting and getting the value of PDF.PDF_RESOURCE_PATH control parameter.

```
// Path to the Resource\PDF directory.
char* szResourcePath = "C:\\PDF\\Resource\\PDF\\";

IG_gctrl_item_set("PDF.PDF_RESOURCE_PATH", AM_TID_MAKELP(AM_TID_CHAR),
szResourcePath, (DWORD)strlen(szResourcePath) + 1, "");
```

```
// Get path to the Resource\PDF directory.
char szResourcePath[_MAX_PATH];

IG_gctrl_item_get("PDF.PDF_RESOURCE_PATH", NULL, (LPVOID)&amp;szResourcePath,
sizeof(szResourcePath) - 1, NULL, NULL, 0, NULL);
```

If the PDF global parameters are not defined, the ImageGear PDF Component behavior depends on the OS:

- Resource content gets from the ImageGear component directory defined by COMM.PATH parameter
- Host fonts get from the Windows font directory
- Temporary directory gets from the GetTempPath() result

1.3.3.1.4 IG_PDF_register_authproc

Registers the authorization callback, which will be called when opening a secured PDF file, i.e., a PDF that has either the user or the master password set.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_register_authproc(
        LPFNIG_PDF_AUTHPROC lpfnAuthProc,
        LPVOID lpAuthData
);
```

Arguments:

Name	Type	Description
lpfnAuthProc	LPFNIG_PDF_AUTHPROC	Authorization callback, called only if the file has been secured (that is, if the file has either the user or the master password set). This callback should obtain whatever information is needed to determine whether the user is authorized to open the file, then call IG_PDF_doc_perm_request (which returns the permissions that the authentication data enables).
lpAuthData	LPVOID	Pointer to user-supplied data to pass to lpfnAuthProc each time it is called.

Return Value:

Error count

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.1.5 IG_PDF_terminate

This function terminates the ImageGear PDF component, and must be called before terminating a user application.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_terminate();
```

Return Value:

Error count

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

For a multi-threaded application, you must call IG_PDF_initialize and IG_PDF_terminate in the main thread and in each worker thread which uses PDF component. See Single- and Multi-Threaded Applications for more information.

1.3.3.1.6 IG_PDF_text_extract

This function extracts text from pages determined by arguments nStartPage and nPageCount.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_text_extract(
        LPSTR lpszFileName,
        LPSTR lpszTextName,
        UINT nStartPage,
        UINT nPageCount
);
```

Arguments:

Name	Type	Description
lpFileName	LPSTR	Name of the PDF or PS document.
lpTextFileName	LPSTR	Name of the output TXT file.
nStartPage	UINT	Number of the first page.
nPageCount	UINT	Total number of pages to be processed (starting at nStartPage).

Return Value:

Error count

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.1.7 IG_PDF_translate_to_host

Translates a string from Unicode or PDFDocEncoding to host encoding.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_translate_to_host(
        LPCSTR szInPDFStr,
        LONG nInPDFStrSize,
        LPSTR szOutHostStr,
        LONG nOutHostStrSize,
        LPLONG lpnOutHostStrBytes
);
```

Arguments:

Name	Type	Description
szInPDFStr	LPCSTR	Pointer to the string to translate (may point to the same memory as szOutHostStr, allowing strings to translate in place).
nInPDFStrSize	LONG	The length of szInPDFStr, in bytes.
szOutHostStr	LPSTR	Pointer to the translated string (may point to the same memory as szInPDFStr).
nOutHostStrSize	LONG	The length of the szOutHostStr buffer, in bytes.
lpnOutHostStrBytes	LPLONG	Number of bytes in the translated string szOutHostStr.

Return Value:

Error count

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function is useful when setting or retrieving displayed text that might be in Unicode, such as text that appears in a text annotation or bookmark.

A character that cannot be converted to the destination encoding is replaced with a space.

Host encoding is a platform-dependent encoding for the host machine. For non-UNIX Roman systems, it is MacRomanEncoding in Mac OS and WinAnsiEncoding in Windows. In UNIX (except HP-UX) Roman systems, it is ISO8859-1 (ISO Latin-1); for HP-UX, it is HP-ROMAN8. See Appendix D in the PDF Reference for descriptions of MacRomanEncoding, WinAnsiEncoding, and PDFDocEncoding.

For non-Roman systems, the host encoding may be a variety of encodings, which are defined by a CMap (character map). See Section 5.6.4 in the PDF Reference for information on CMaps.

Use IG_PDF_get_host_encoding to determine if a system's host encoding is Roman or not.

1.3.3.1.8 IG_PDF_translate_to_pdf

Translates a string from host encoding to PDFDocEncoding or Unicode.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_translate_to_pdf(
        AT_PDF_BOOL bUseUnicode,
        LPCSTR szInHostStr,
        LONG nInHostStrSize,
        LPSTR szOutPDFStr,
        LONG nOutPDFStrSize,
        LPLONG lpnOutPDFStrBytes
);
```

Arguments:

Name	Type	Description
bUseUnicode	AT_PDF_BOOL	If TRUE, translate the string to Unicode; otherwise use PDFDocEncoding.
szInHostStr	LPCSTR	Pointer to the string to translate (may point to the same memory as szOutPDFStr, allowing strings to translate in place).
nInHostStrSize	LONG	Number of bytes in szOutPDFStr.
szOutPDFStr	LPSTR	Pointer to the translated string (may point to the same memory as szInHostStr).
nOutPDFStrSize	LONG	The length of the szOutPDFStr buffer, in bytes.
lpnOutPDFStrBytes	LPLONG	Number of bytes in the translated string szOutPDFStr.

Return Value:

Error count

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function is useful when using text that must be in PDFDocEncoding or Unicode, such as text in a text annotation, bookmark, or article title.

A character that cannot be converted to the destination encoding is replaced with a space.

For example, it converts \n to a space character (\r is present in PDFDocEncoding and is left unchanged).

Host encoding is a platform-dependent encoding for the host machine. For non-UNIX Roman systems, it is MacRomanEncoding in Mac OS and WinAnsiEncoding in Windows. In UNIX (except HP-UX) Roman systems, it is ISO8859-1 (ISO Latin-1); for HP-UX, it is HP-ROMAN8. See Appendix D in the PDF Reference for descriptions of MacRomanEncoding, WinAnsiEncoding, and PDFDocEncoding.

For non-Roman systems, the host encoding may be a variety of encodings, which are defined by a CMap (character map). See Section 5.6.4 in the PDF Reference for a list of predefined CMaps.

Use IG_PDF_get_host_encoding to determine if a system's host encoding is Roman or not.

1.3.3.2 PDF Component Callback Functions Reference

This section provides information about the Callback functions.

- LPFNIG_PDF_AUTHPROC
- LPFNIG_PDF_STREAM_PROC
- LPFNIG_PDF_STREAM_DESTROYPROC
- LPFNIG_PDF_SYSFONT_ENUMPROC
- LPFNIG_PDE_CLIP_ENUMPROC

1.3.3.2.1 LPFNIG_PDF_AUTHPROC

This callback is used by document open routine; it is called when an encrypted document is being opened to determine whether or not the user is authorized to open the file.

Declaration:

```
typedef AT_PDF_BOOL (LPACCUAPI LPFNIG_PDF_AUTHPROC)(
        HIG_PDF_DOC hDoc,
        LPVOID clientData
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The PDF document to open.
clientData	LPVOID	User-supplied data that was passed in the call to IG_PDF_register_authproc.

Return Value:

TRUE if the user is authorized to open the document; FALSE otherwise.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This callback implements whatever authorization strategy you choose and calls the callbacks of the appropriate security handler (the one that was used to secure the document) to obtain and check authorization data.

The LPFNIG_PDF_AUTHPROC should obtain the authorization data (a password) and call IG_PDF_doc_perm_request(). IG_PDF_doc_perm_request in turn calls the document encryption handler's Authorize function, which returns the permissions that the authorization data enables. IG_PDF_doc_perm_request adds these permissions to those currently allowed, and returns the new set of allowed permissions.

1.3.3.2.2 LPFNIG_PDF_STREAM_PROC

Callback for use by IG_PDF_stream_read_CB_register and IG_PDF_stream_write_CB_register.

Declaration:

```
typedef LONG (LPACCUAPI LPFNIG_PDF_STREAM_PROC)(
        LPSTR lpData,
        UINT nDataLen,
        LPVOID clientData
);
```

Arguments:

Name	Type	Description
lpData	LPSTR	Buffer into which your procedure must place the number of bytes specified by nDataLen.
nDataLen	UINT	Number of bytes to read from the stream and place into data.
clientData	LPVOID	User-supplied data that was specified in the call to IG_PDF_stream_read_CB_register or IG_PDF_stream_write_CB_register.

Return Value:

Returns the number of bytes actually read or written.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This procedure must return the number of bytes specified by nDataLen, obtaining them in any way it wishes.

1.3.3.2.3 LPFNIG_PDF_STREAM_DESTROYPROC

Callback for use by IG_PDF_stream_write_CB_register.

Declaration:

```
typedef void (LPACCUAPI LPFNIG_PDF_STREAM_DESTROYPROC)(
        LPVOID clientData
);
```

Arguments:

Name	Type	Description
clientData	LPVOID	User-supplied data that was specified in the call to IG_PDF_stream_write_CB_register.

Return Value:

Returns the number of bytes actually read or written.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This callback function is called at end of stream so you can perform clean up and free allocated memory.

1.3.3.2.4 LPFNIG_PDF_SYSFONT_ENUMPROC

This callback for IG_PDF_sysfont_enumerate is called once for each system font.

Declaration:

```
typedef (LPACCUAPI LPFNIG_PDF_SYSFONT_ENUMPROC)(
        HIG_PDF_SYSFONT hSysFont,
        LPVOID clientData
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	The system font.
clientData	LPVOID	User-supplied data that was specified in the call to IG_PDF_sysfont_enumerate.

Return Value:

Returns TRUE to continue enumeration; FALSE to halt enumeration.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.2.5 LPFNIG_PDE_CLIP_ENUMPROC

Callback for IG_PDE_clip_enumerate_elements(), which enumerates all of a PDE Clip's PDE Elements in a flattened manner.

Declaration:

```
typedef AT_PDF_BOOL (LPACCUAPI LPFNIG_PDE_CLIP_ENUMPROC)(
        HIG_PDE_ELEMENT hElement,
        LPVOID clientData
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	The PDE Element currently being enumerated.
clientData	LPVOID	User-supplied data that was passed in the call to IG_PDE_clip_enumerate_elements.

Return Value:

If FALSE, enumeration halts. If TRUE, enumeration continues.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.3 PDF Component Macros Reference

This section provides information about the PDF Macros.

- AM_PDF_LONG_TO_FIXED
- AM_PDF_FIXED_ROUND_TO_LONG
- AM_PDF_FIXED_TRUNC_TO_LONG
- AM_PDF_SHORT_TO_FIXED
- AM_PDF_FIXED_ROUND_TO_SHORT
- AM_PDF_FIXED_TRUNC_TO_SHORT
- AM_PDF_DOUBLE_TO_FIXED
- AM_PDF_FIXED_TO_DOUBLE

1.3.3.3.1 AM_PDF_LONG_TO_FIXED

Converts x to AT_PDF_FIXED and returns it.

Declaration:

```
AM_PDF_LONG_TO_FIXED(x)
```

Arguments:

X	Long integer value.

Return Value:

AT_PDF_FIXED value

1.3.3.3.2 AM_PDF_FIXED_ROUND_TO_LONG

Converts the AT_PDF_FIXED number f to an integer, rounding it to the nearest long integer value and returns it.

Declaration:

```
AM_PDF_FIXED_ROUND_TO_LONG (f)
```

Arguments:

f	Fixed value.

Return Value:

Long integer value

1.3.3.3.3 AM_PDF_FIXED_TRUNC_TO_LONG

Converts the AT_PDF_FIXED number f to an integer, truncating it to the next lower long integer value and returns it.

Declaration:

```
AM_PDF_FIXED_TRUNC_TO_LONG (f)
```

Arguments:

f	Fixed value.

Return Value:

Long integer value

1.3.3.3.4 AM_PDF_SHORT_TO_FIXED

Converts x to AT_PDF_FIXED and returns it.

Declaration:

```
AM_PDF_SHORT_TO_FIXED(x)
```

Arguments:

x	Short integer value.

Return Value:

AT_PDF_FIXED value

1.3.3.3.5 AM_PDF_FIXED_ROUND_TO_SHORT

Converts the AT_PDF_FIXED number f to an integer, rounding it to the nearest short integer value and returns it.

Declaration:

```
AM_PDF_FIXED_ROUND_TO_SHORT (f)
```

Arguments:

f	Fixed value.

Return Value:

Short integer value

1.3.3.3.6 AM_PDF_FIXED_TRUNC_TO_SHORT

Converts the AT_PDF_FIXED number f to an integer, truncating it to the next lower short integer value and returns it.

Declaration:

```
AM_PDF_FIXED_TRUNC_TO_SHORT (f)
```

Arguments:

f	Fixed value.

Return Value:

Short integer value

1.3.3.3.7 AM_PDF_DOUBLE_TO_FIXED

Converts x to AT_PDF_FIXED and returns it.

Declaration:

```
AM_PDF_DOUBLE_TO_FIXED (x)
```

Arguments:

x	Double value.

Return Value:

AT_PDF_FIXED Value

1.3.3.3.8 AM_PDF_FIXED_TO_DOUBLE

Converts the AT_PDF_FIXED number f to double value and returns it.

Declaration:

```
AM_PDF_FIXED_TO_DOUBLE (f)
```

Arguments:

f	Fixed value.

Return Value:

Double Value

1.3.3.4 PDF Component Objects Reference

This section provides information about the PDF Objects, grouped as follows:

- Basic Objects
- General Objects
- Page Editing Objects and Elements

1.3.3.4.1 Basic Objects

This section describes the basic PDF objects and utility functions used throughout the ImageGear PDF API. These objects provide access to the building blocks used to construct PDF documents. Its functions allow applications to manipulate the low-level data in a PDF file, such as strings, numbers, and dictionaries. Adobe PDF supports the following basic platform-independent types of object:

- Arrays
- Boolean values
- Dictionaries
- Integer and Fixed (real) numbers
- Names
- The null object
- Strings

These objects may be labeled so that they can be referred to by other objects. A labeled object is called an indirect object. When a direct object is created, the object itself is returned. As a result, a direct object can only be attached to one other Base object at a time; it cannot, for example, be shared by two different dictionaries. When an indirect object is created, something equivalent to a pointer to the object is returned. As a result, an indirect object can be attached to multiple places in a PDF file simultaneously; it can, for example, be shared by two different dictionaries.

PDF documents are trees of these Base objects. Base objects represent document components such as bookmarks, pages, and fonts. Unlike using the other ImageGear PDF Objects functions, using Base Object functions improperly could result in an invalid PDF file. Therefore, you should not use Base Object methods unless necessary, for example to add private data to portions of a PDF file that cannot be accessed in other ways.

The following table describes the objects supported by the ImageGear PDF component:

Basic Objects

HIG_PDF_BASOBJ	Basic Object - basic PDF object interface. PDF supports eight basic types of object: • Arrays • Boolean values • Dictionaries • Integer and Fixed (real) numbers • Names • Streams • The null object • Strings Objects may be labeled so that they can be referred to by other objects. A labeled object is called an indirect object.
HIG_PDF_BASARR	Basic Array - an array object is a one-dimensional collection of objects arranged sequentially. Unlike arrays in many other computer languages, PDF arrays may be heterogeneous; that is, an array's elements may be any combination of numbers, strings, dictionaries, or any other objects, including other arrays.
HIG_PDF_BASBOOL	Basic Boolean - PDF provides Boolean objects identified by the keywords TRUE and FALSE. Boolean objects can be used as the values of array elements and dictionary entries.
HIG_PDF_BASDICT	Basic Dictionary - a dictionary object is an associative table containing pairs of objects, known as the dictionary's entries. The first element of each entry is the key and the second element is the value. The key must be a name. The value can be any kind of object, including another dictionary.
HIG_PDF_BASFIXED	Basic Fixed - Fixed objects approximate mathematical real numbers, but with limited range and precision; they are typically represented in fixed-point, rather than floating-point, form.
HIG_PDF_BASINT	Basic Integer - Integer objects represent mathematical integers within a certain interval centered at 0.
HIG_PDF_BASNAME	Basic Name - A name object is an atomic symbol uniquely defined by a sequence of characters. Uniquely defined means that any two name objects made up of the same sequence of characters are identically the same object. Atomic means that a name has no internal structure; although it is defined by a sequence of characters, those characters are not "elements" of the name.
HIG_PDF_BASNULL	Basic Null - The null object has a type and value that are unequal to those of any other object. There is only one object of type null, denoted by the keyword null.

1.3.3.4.1.1 HIG_PDF_BASOBJ

Handle to the basic object.

Members:

IG_PDF_basobj_get_type	Gets an object's type.
IG_PDF_basobj_release	Releases PDF Object.
IG_PDF_basobj_remove	Removes basic object.

1.3.3.4.1.1.1 IG_PDF_basobj_get_type

Gets an object's type.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basobj_get_type (
        HIG_PDF_BASOBJ hObject,
        LPLONG lpnType
);
```

Arguments:

Name	Type	Description
hObject	HIG_PDF_BASOBJ	Basic object.
lpnType	LPLONG	The object's type.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.1.2 IG_PDF_basobj_release

Releases PDF Object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basobj_release (
        HIG_PDF_BASOBJ hObject
);
```

Arguments:

Name	Type	Description
hObject	HIG_PDF_BASOBJ	Object to release.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.1.3 IG_PDF_basobj_remove

Removes basic object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basobj_remove(
        HIG_PDF_BASOBJ hObject
);
```

Arguments:

Name	Type	Description
hObject	HIG_PDF_BASOBJ	The object to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If a composite object (array, dictionary, or stream) is removed, all the direct objects in it are automatically removed, but the indirect objects in it are not removed.

1.3.3.4.1.2 HIG_PDF_BASARR

General HIG_PDF_BASOBJ is used to handle to the basic array object. An array object is a one-dimensional collection of objects arranged sequentially. Unlike arrays in many other computer languages, PDF arrays may be heterogeneous; that is, an array's elements may be any combination of numbers, strings, dictionaries, or any other objects, including other arrays.

Members:

IG_PDF_basarr_create	Creates a new array of objects.
IG_PDF_basarr_get_length	Gets the number of elements in hArray.
IG_PDF_basarr_get	Gets the specified element from an array.
IG_PDF_basarr_put	Puts the specified object into the specified location in an array.
IG_PDF_basarr_put_int	Puts the specified fixed value into the specified location in an array.
IG_PDF_basarr_put_fixed	Puts the specified fixed value into the specified location in an array.
IG_PDF_basarr_put_bool	Puts the specified Boolean value into the specified location in an array.
IG_PDF_basarr_put_name	Puts the specified name value into the specified location in an array.
IG_PDF_basarr_remove	Finds the first element, if any, equal to the specified object and removes it from the array.
IG_PDF_basarr_remove_nth	Checks whether the position is within the array bounds and then removes it from the array and moves each subsequent element to the slot with the next smaller index and decrements the array's length by 1.

1.3.3.4.1.2.1 IG_PDF_basarr_create

Creates a new array of objects.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_create(
        HIG_PDF_DOC hDoc,
        AT_PDF_BOOL bIndirect,
        UINT nElements,
        LPHIG_PDF_BASOBJ lphArray
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the array is used.
bIndirect	AT_PDF_BOOL	If TRUE, creates the array as an indirect object. If FALSE, creates the dictionary as a direct object.
nElements	UINT	Number of entries in the array. This value is only a hint - the arrays grow dynamically as needed.
lphArray	LPHIG_PDF_BASOBJ	New array.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.2.2 IG_PDF_basarr_get_length

Gets the number of elements in hArray.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_get_length(
        HIG_PDF_BASOBJ hArray,
        LPLONG lpnLength
);
```

Arguments:

Name	Type	Description
hArray	HIG_PDF_BASOBJ	The array.
lpnLength	LPLONG	The number of elements in hArray.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.2.3 IG_PDF_basarr_get

Gets the specified element from an array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_get(
        HIG_PDF_BASOBJ hArray,
        UINT nIndex,
        LPHIG_PDF_BASOBJ lphObject
);
```

Arguments:

Name	Type	Description
hArray	HIG_PDF_BASOBJ	The array from which an element is obtained.
nIndex	UINT	The array element to obtain. The first element in an array has an index of zero.
lphObject	LPHIG_PDF_BASOBJ	The basic object occupying the nIndex element of array. Returns IG_PDF_BASIC_NULL object if index is outside the array bounds.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.2.4 IG_PDF_basarr_put

Puts the specified object into the specified location in an array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put(
        HIG_PDF_BASOBJ hArray,
        UINT nIndex,
        HIG_PDF_BASOBJ hObject
);
```

Arguments:

Name	Type	Description
hArray	HIG_PDF_BASOBJ	The array in which hObject is stored.
nIndex	UINT	The location in array to store hObject. The first element of an array has an index of zero.
hObject	HIG_PDF_BASOBJ	The object to insert into hArray.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The array is extended as much as necessary.

1.3.3.4.1.2.5 IG_PDF_basarr_put_int

Puts the specified integer value into the specified location in an array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put_int(
        HIG_PDF_BASOBJ hArray,
        UINT nIndex,
        AT_PDF_BOOL bIndirect,
        INT nValue
);
```

Arguments:

Name	Type	Description
hArray	HIG_PDF_BASOBJ	The array in which a value is stored.
nIndex	UINT	The location in array to store a value. The first element of an array has an index of zero.
bIndirect	AT_PDF_BOOL	If TRUE, creates the key value as an indirect object. If FALSE, creates the key value as a direct object.
nValue	INT	The integer value to insert into hArray.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The array is extended as much as necessary.

1.3.3.4.1.2.6 IG_PDF_basarr_put_fixed

Puts the specified fixed value into the specified location in an array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put_fixed(
        HIG_PDF_BASOBJ hArray,
        UINT nIndex,
        AT_PDF_BOOL bIndirect,
        AT_PDF_FIXED nValue
);
```

Arguments:

Name	Type	Description
hArray	HIG_PDF_BASOBJ	The array in which a value is stored.
nIndex	UINT	The location in array to store a value. The first element of an array has an index of zero.
bIndirect	AT_PDF_BOOL	If TRUE, creates the key value as an indirect object. If FALSE, creates the key value as a direct object.
nValue	AT_PDF_FIXED	The fixed value to insert into hArray.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The array is extended as much as necessary.

1.3.3.4.1.2.7 IG_PDF_basarr_put_bool

Puts the specified Boolean value into the specified location in an array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put_bool(
        HIG_PDF_BASOBJ hArray,
        UINT nIndex,
        AT_PDF_BOOL bIndirect,
        AT_PDF_BOOL bValue
);
```

Arguments:

Name	Type	Description
hArray	HIG_PDF_BASOBJ	The array in which a value is stored.
nIndex	UINT	The location in array to store a value. The first element of an array has an index of zero.
bIndirect	AT_PDF_BOOL	If TRUE, creates the key value as an indirect object. If FALSE, creates the key value as a direct object.
bValue	AT_PDF_BOOL	The Boolean value to insert into hArray.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The array is extended as much as necessary.

1.3.3.4.1.2.8 IG_PDF_basarr_put_name

Puts the specified name value into the specified location in an array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_put_name(
        HIG_PDF_BASOBJ hArray,
        UINT nIndex,
        AT_PDF_BOOL bIndirect,
        HIG_PDF_ATOM nName
);
```

Arguments:

Name	Type	Description
hArray	HIG_PDF_BASOBJ	The array in which a value is stored.
nIndex	UINT	The location in array to store a value. The first element of an array has an index of zero.
bIndirect	AT_PDF_BOOL	If TRUE, creates the key value as an indirect object. If FALSE, creates the key value as a direct object.
nName	HIG_PDF_ATOM	The name value to insert into hArray.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The array is extended as much as necessary.

1.3.3.4.1.2.9 IG_PDF_basarr_remove

Finds the first element, if any, equal to the specified object and removes it from the array.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_remove(
        HIG_PDF_BASOBJ hArray,
        HIG_PDF_BASOBJ hObject
);
```

Arguments:

Name	Type	Description
hArray	HIG_PDF_BASOBJ	The array in which hObject is removed.
hObject	HIG_PDF_BASOBJ	The object to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.2.10 IG_PDF_basarr_remove_nth

Checks whether the position is within the array bounds, then removes it from the array, moves each subsequent element to the slot with the next smaller index, and decrements the array's length by 1.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basarr_remove_nth(
        HIG_PDF_BASOBJ hArray,
        UINT nIndex
);
```

Arguments:

Name	Type	Description
hArray	HIG_PDF_BASOBJ	The array from which to remove the member.
nIndex	UINT	The index for the array member to remove. Array indices start at 0.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.3 HIG_PDF_BASBOOL

Handle to the basic Boolean object. PDF provides Boolean objects identified by the keywords TRUE and FALSE. Boolean objects can be used as the values of array elements and dictionary entries.

Members:

IG_PDF_basbool_create	Creates a new Boolean object associated with the specified document and having the specified value.
IG_PDF_basbool_get_value	Gets the value of the specified Boolean object.

1.3.3.4.1.3.1 IG_PDF_basbool_create

Creates a new Boolean object associated with the specified document and having the specified value.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basbool_create(
        HIG_PDF_DOC hDoc,
        AT_PDF_BOOL bIndirect,
        AT_PDF_BOOL bValue,
        LPHIG_PDF_BASOBJ lphBool
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the Boolean is used.
bIndirect	AT_PDF_BOOL	If TRUE, creates the Boolean object as an indirect object. If FALSE, creates the Boolean as a direct object.
bValue	AT_PDF_BOOL	The value the new Boolean will have.
lphBool	LPHIG_PDF_BASOBJ	A Boolean object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.3.2 IG_PDF_basbool_get_value

Gets the value of the specified Boolean object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basbool_get_value (
        HIG_PDF_BASOBJ hBool,
        LPAT_PDF_BOOL lpbValue
);
```

Arguments:

Name	Type	Description
hBool	LPHIG_PDF_BASOBJ	Object.
lpbValue	LPAT_PDF_BOOL	Value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.4 HIG_PDF_BASDICT

General HIG_PDF_BASOBJ is used as a handle to the basic dictionary object. A dictionary object is an associative table containing pairs of objects, known as the dictionary's entries. The first element of each entry is the key, and the second element is the value. The key must be a name. The value can be any kind of object, including another dictionary.

Members:

IG_PDF_basdict_create	Creates a new dictionary.
IG_PDF_basdict_known	Tests whether a specific key is found in the specified dictionary.
IG_PDF_basdict_get	Gets the value of the specified key in the specified dictionary.
IG_PDF_basdict_put	Sets the value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_put_int	Sets the integer value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_put_fixed	Sets the Boolean value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_put_bool	Sets the Boolean value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_put_name	Sets the name value of a dictionary key, adding the key to the dictionary if it is not already present.
IG_PDF_basdict_remove	Removes a key-value pair from a dictionary.

1.3.3.4.1.4.1 IG_PDF_basdict_create

Creates a new dictionary.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_create(
        HIG_PDF_DOC hDoc,
        AT_PDF_BOOL bIndirect,
        UINT nEntries,
        LPHIG_PDF_BASOBJ lphDictionary
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the dictionary is used.
bIndirect	AT_PDF_BOOL	If TRUE, creates the dictionary as an indirect object. If FALSE, creates the dictionary as a direct object.
nEntries	UINT	Number of entries in the dictionary. This value is only a hint - the dictionaries grow dynamically as needed.
lphDictionary	LPHIG_PDF_BASOBJ	New dictionary.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

See the PDF Reference for information on dictionary objects that are part of standard PDF, such as annotations or page objects.

1.3.3.4.1.4.2 IG_PDF_basdict_known

Tests whether a specific key is found in the specified dictionary.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_known(
        HIG_PDF_BASOBJ hDictionary,
        HIG_PDF_ATOM hKey,
        LPAT_PDF_BOOL lpbKnown
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_BASOBJ	The dictionary in which to look for key.
hKey	HIG_PDF_ATOM	The key to find. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
lpbKnown	LPAT_PDF_BOOL	TRUE if the value of a key is known (exists and is not null) in hDictionary; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.4.3 IG_PDF_basdict_get

Gets the value of the specified key in the specified dictionary.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_get(
        HIG_PDF_BASOBJ hDictionary,
        HIG_PDF_ATOM hKey,
        LPHIG_PDF_BASOBJ lphObject
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_BASOBJ	The dictionary or stream from which a value is obtained.
hKey	HIG_PDF_ATOM	The key whose value is obtained. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
lphObject	LPHIG_PDF_BASOBJ	The object associated with the specified key. If key is not present or if its value is null, returns an object of type IG_PDF_BASIC_NULL.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If called with a stream object instead of a dictionary object, this function gets the value of the specified key from the stream's attributes dictionary.

1.3.3.4.1.4.4 IG_PDF_basdict_put

Sets the value of a dictionary key, adding the key to the dictionary if it is not already present.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put(
        HIG_PDF_BASOBJ hDictionary,
        HIG_PDF_ATOM hKey,
        HIG_PDF_BASOBJ hObject
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_BASOBJ	The dictionary in which a value is set.
hKey	HIG_PDF_ATOM	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
hObject	HIG_PDF_BASOBJ	The value to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.4.5 IG_PDF_basdict_put_int

Sets the integer value of a dictionary key, adding the key to the dictionary if it is not already present.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put_int(
        HIG_PDF_BASOBJ hDictionary,
        HIG_PDF_ATOM hKey,
        AT_PDF_BOOL bIndirect,
        INT nValue
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_BASOBJ	The dictionary in which a value is set.
hKey	HIG_PDF_ATOM	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
bIndirect	AT_PDF_BOOL	If TRUE, creates the key value as an indirect object. If FALSE, creates the key value as a direct object.
nValue	INT	The integer value to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.4.6 IG_PDF_basdict_put_fixed

Sets the fixed value of a dictionary key, adding the key to the dictionary if it is not already present.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put_fixed(
        HIG_PDF_BASOBJ hDictionary,
        HIG_PDF_ATOM hKey,
        AT_PDF_BOOL bIndirect,
        AT_PDF_FIXED nValue
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_BASOBJ	The dictionary in which a value is set.
hKey	HIG_PDF_ATOM	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
bIndirect	AT_PDF_BOOL	If TRUE, creates the key value as an indirect object. If FALSE, creates the key value as a direct object.
nValue	AT_PDF_FIXED	The fixed value to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.4.7 IG_PDF_basdict_put_bool

Sets the Boolean value of a dictionary key, adding the key to the dictionary if it is not already present.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put_bool(
        HIG_PDF_BASOBJ hDictionary,
        HIG_PDF_ATOM hKey,
        AT_PDF_BOOL bIndirect,
        AT_PDF_BOOL bValue
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_BASOBJ	The dictionary in which a value is set.
hKey	a href="IGDLL-26-079.html">HIG_PDF_ATOM	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
bIndirect	AT_PDF_BOOL	If true, creates the key value as an indirect object. If FALSE, creates the key value as a direct object.
bValue	AT_PDF_BOOL	The Boolean value to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.4.8 IG_PDF_basdict_put_name

Sets the name value of a dictionary key, adding the key to the dictionary if it is not already present.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_put_name(
        HIG_PDF_BASOBJ hDictionary,
        HIG_PDF_ATOM hKey,
        AT_PDF_BOOL bIndirect,
        HIG_PDF_ATOM nName
);
```

Arguments:

Name	**Type**	**Description**
hDictionary	HIG_PDF_BASOBJ	The dictionary in which a value is set.
hKey	HIG_PDF_ATOM	The key whose value is set. See the PDF Reference to obtain the names of keys in dictionary objects that are part of standard PDF, such as annotations or page objects.
bIndirect	AT_PDF_BOOL	If TRUE, creates the key value as an indirect object. If FALSE, creates the key value as a direct object.
nName	HIG_PDF_ATOM	The name value to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.4.9 IG_PDF_basdict_remove

Removes a key-value pair from a dictionary.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basdict_remove(
        HIG_PDF_BASOBJ hDictionary,
        HIG_PDF_ATOM hKey
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_BASOBJ	The dictionary in which a key is removed.
hKey	HIG_PDF_ATOM	The key to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.5 HIG_PDF_BASFIXED

Handle to the basic fixed object. Fixed objects approximate mathematical real numbers, but with limited range and precision; they are typically represented in fixed-point, rather than floating-point, form.

Members:

IG_PDF_basfixed_create	Creates a new fixed object associated with the specified document and having the specified value.
IG_PDF_basfixed_get_value	Gets the value of the specified fixed object.

1.3.3.4.1.5.1 IG_PDF_basfixed_create

Creates a new fixed object associated with the specified document and having the specified value.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basfixed_create(
        HIG_PDF_DOC hDoc,
        AT_PDF_BOOL bIndirect,
        LONG nValue,
        LPHIG_PDF_BASOBJ lphFixed
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the object is used.
bIndirect	AT_PDF_BOOL	If TRUE, creates the fixed as an indirect object. If FALSE, creates the fixed as a direct object.
nValue	LONG	The value the new fixed will have.
lphFixed	LPHIG_PDF_BASOBJ	The value the new fixed will have.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.5.2 IG_PDF_basfixed_get_value

Gets the value of the specified fixed object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basfixed_get_value (
        HIG_PDF_BASOBJ hFixed,
        LPLONG lpnValue
);
```

Arguments:

Name	Type	Description
hFixed	HIG_PDF_BASOBJ	Object.
lpnValue	LPLONG	Value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.6 HIG_PDF_BASINT

Handle to the basic integer object. Integer objects represent mathematical integers within a certain interval centered at 0.

Members:

IG_PDF_basint_create	Creates a new integer object associated with the specified document and having the specified value.
IG_PDF_basint_get_value	Gets the value of the specified integer object.

1.3.3.4.1.6.1 IG_PDF_basint_create

Creates a new integer object associated with the specified document and having the specified value.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basint_create(
        HIG_PDF_DOC hDoc,
        AT_PDF_BOOL bIndirect,
        LONG nValue,
        LPHIG_PDF_BASOBJ lphInt
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the object is used.
bIndirect	AT_PDF_BOOL	If TRUE, creates the integer as an indirect object. If FALSE, creates the integer as a direct object.
nValue	LONG	The value the new integer will have.
lphInt	LPHIG_PDF_BASOBJ	An integer object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.6.2 IG_PDF_basint_get_value

Gets the value of the specified integer object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basint_get_value (
        HIG_PDF_BASOBJ hInt,
        LPLONG lpnValue
);
```

Arguments:

Name	Type	Description
hInt	HIG_PDF_BASOBJ	Object.
lpnValue	LPLONG	Value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.7 HIG_PDF_BASNAME

Handle to the basic name object. A name object is an atomic symbol uniquely defined by a sequence of characters.

Uniquely defined means that any two name objects made up of the same sequence of characters are identically the same object. Atomic means that a name has no internal structure; although it is defined by a sequence of characters, those characters are not "elements" of the name.

Members:

IG_PDF_basname_create	Creates a new name object associated with the specified document and having the specified value.
IG_PDF_basname_get_value	Gets the value of the specified name object.

1.3.3.4.1.7.1 IG_PDF_basname_create

Creates a new name object associated with the specified document and having the specified value.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basname_create(
        HIG_PDF_DOC hDoc,
        AT_PDF_BOOL bIndirect,
        HIG_PDF_ATOM hNameVal,
        LPHIG_PDF_BASOBJ lphName
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the object is used.
bIndirect	AT_PDF_BOOL	If TRUE, creates the name as an indirect object. If FALSE, creates the name as a direct object.
hNameVal	HIG_PDF_ATOM	The value the new name will have.
lphName	LPHIG_PDF_BASOBJ	A name object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.7.2 IG_PDF_basname_get_value

Gets the value of the specified name object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basname_get_value (
        HIG_PDF_BASOBJ hName,
        LPHIG_PDF_ATOM lphNameVal
);
```

Arguments:

Name	Type	Description
hName	HIG_PDF_BASOBJ	Object.
lpnNameVal	LPHIG_PDF_ATOM	Value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.8 HIG_PDF_BASNULL

Handle to the basic null object. The null object has a type and value that are unequal to those of any other object. There is only one object of type null, denoted by the keyword null.

Members:

IG_PDF_basnull_create	Creates a direct null object.

1.3.3.4.1.8.1 IG_PDF_basnull_create

Creates a direct null object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basnull_create(
        LPHIG_PDF_BASOBJ lphNull
);
```

Arguments:

Name	Type	Description
lphNull	LPHIG_PDF_BASOBJ	Null object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.9 HIG_PDF_BASSTR

Handle to the basic string object. A string object consists of a series of bytes-unsigned integer values in the range 0 to 255. The string elements are not integer objects, but are stored in a more compact format.

Members:

IG_PDF_basstr_create	Creates a new string object associated with the specified document and having the specified value.
IG_PDF_basstr_get_value	Copies at most nLen bytes from obj's string value into lpString, and stores the actual length of the basic string in lpnBytes.

1.3.3.4.1.9.1 IG_PDF_basstr_create

Creates a new string object associated with the specified document and having the specified value.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basstr_create(
        HIG_PDF_DOC hDoc,
        AT_PDF_BOOL bIndirect,
        LPBYTE lpString,
        LONG nBytes,
        LPHIG_PDF_BASOBJ lphString
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the object is used.
bIndirect	AT_PDF_BOOL	If TRUE, creates the string as an indirect object. If FALSE, creates the string as a direct object.
lpString	LPBYTE	The value the new string will have.
nBytes	LONG	The length of lpString.
lphString	LPHIG_PDF_BASOBJ	The value the new string will have.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.9.2 IG_PDF_basstr_get_value

Copies at most nLen bytes from obj's string value into lpString, and stores the actual length of the basic string in lpnBytes.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_basstr_get_value (
        HIG_PDF_BASOBJ hString,
        LPBYTE lpString,
        LONG nLen,
        LPLONG lpnBytes
);
```

Arguments:

Name	Type	Description
hString	HIG_PDF_BASOBJ	Object.
lpString	LPBYTE	The buffer into which the original string value is copied or NULL.
nLen	LONG	The length of buffer or 0.
lpnBytes	LPLONG	The length of the original string in bytes. Must be a non-NULL pointer.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.1.10 HIG_PDF_BASSTREAM

Handle to the basic stream object.

Members:

IG_PDF_basstream_create	Creates a new stream object associated with the specified document and having the specified value.
IG_PDF_basstream_get_dict	Gets a stream's attributes dictionary.
IG_PDF_basstream_get_value	Copies at most nBufferLenbytes from object's stream value into lpBuffer, and stores the actual length of the basic string in lpnStreamLen.

1.3.3.4.1.10.1 IG_PDF_basstream_create

Creates a new stream object associated with the specified document and having the specified value.

Declaration:

```
AT_ERRCOUNTACCUAPI IG_PDF_basstream_create(
        HIG_PDF_DOC hDoc,
        HIG_PDF_STREAM hStream,
        AT_PDF_BOOL bEncodeTheSourceData,
        HIG_PDF_BASOBJ hAttributesDictionary,
        HIG_PDF_BASOBJ hEncodeParameters,
        LPHIG_PDF_BASOBJ lphBasStream
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the object is used.
hStream	HIG_PDF_STREAM	The source stream containing the data to copy into the new stream.
bEncodeTheSourceData	AT_PDF_BOOL	Determines whether the data in stm should be encoded using filters specified in hAttributesDictionary.
hAttributesDictionary	HIG_PDF_BASOBJ	Either the NULL, or a dictionary containing stream attributes, such as the length of the stream data and a list of decoding filters, as defined in Section 3.2.7 in the PDF Reference.
hEncodeParameters	HIG_PDF_BASOBJ	The parameters to be used by the encoding filters.
lphBasStream	LPHIG_PDF_BASOBJ	The value the new stream will have.

Return Value:

Error count.

1.3.3.4.1.10.2 IG_PDF_basstream_get_dict

Gets a stream’s attributes dictionary.

Declaration:

```
AT_ERRCOUNTACCUAPI IG_PDF_basstream_get_dict(
        HIG_PDF_BASOBJ hStream,
        LPHIG_PDF_BASOBJ lphAttrDictionary
);
```

Arguments:

Name	Type	Description
hStream	HIG_PDF_BASOBJ	The Basic stream object.
lphAttrDictionary	LPHIG_PDF_BASOBJ	The stream’s attributes dictionary.

Return Value:

Error count.

1.3.3.4.1.10.3 IG_PDF_basstream_get_value

Copies at most nBufferLenbytes from object's stream value into lpBuffer, and stores the actual length of the basic string in lpnStreamLen.

Declaration:

```
AT_ERRCOUNTACCUAPI IG_PDF_basstream_get_value(
        HIG_PDF_BASOBJ hStream,
        LPBYTE lpBuffer,
        LONG nBufferLen,
        LPLONG lpnStreamLen
);
```

Arguments:

Name	**Type**	**Description**
hStream	HIG_PDF_BASOBJ	The Basic stream object.
lpBuffer	LPBYTE	The buffer into which the original stream content is copied or NULL.
nBufferLen	LONG	The length of buffer or 0.
lpnStreamLen	LPLONG	The length of the original stream in bytes. Must be a non-NULL pointer.

Return Value:

Error count.

1.3.3.4.2 General Objects

This section describes a group of objects that provide access to PDF document's components such as metadata, pages, fonts, etc. These objects and functions allow applications to manipulate the PDF content and data. Some of the objects allow you to work with host system fonts and encodings, and the supplementary objects such as Atom and Stream used to simplify and optimize working with PDF content.

The following table describes the general objects supported by the ImageGear PDF component:

General Objects	
HIG_PDF_ACTION	Handle to a PDF action object, which is a task that is performed when a user clicks on a link or a bookmark.
HIG_PDF_ATOM	Atom - a hashed token used in place of strings to optimize performance (it is much faster to compare Atoms than strings). Many functions use Atoms.
HIG_PDF_BOOKMARK	Handle to a PDF bookmark object, which allows the user to navigate interactively from one part of the document to another.
HIG_PDF_DESTINATION	Handle to a PDF destination object, which represents a particular view of a page in a document.
HIG_PDF_DOC	Document - the underlying PDF representation of a document. Through PDF Document, your application can perform most of the Edit Pages operations (delete, replace, and so on). Thumbnails can be created and deleted through this object. You can set and retrieve document information fields through this object as well.
HIG_PDF_PAGE	Page - a single page in the PDF representation of a document. A page contains a series of objects representing the objects drawn on the page (Graphic), a list of resources used in drawing the page, annotations (Annotation), an optional thumbnail image of the page, and the beads used in any articles that occur on the page.
HIG_PDF_STREAM	Stream - a data stream that may be a buffer in memory, or an arbitrary user-written procedure. Typically used to extract or provide data.
HIG_PDF_STYLE	Style - provides access to information about the fonts, font sizes, and colors used in a Word.
HIG_PDF_SYSENCODING	SysEncoding - provides system encoding for a PDF file.
HIG_PDF_SYSFONT	SysFont - a reference to a font installed in the host system. SysFont methods allow you to list the fonts available in the host system and to find a font in the system that matches a PDE Font, if it is present.
HIG_PDF_WORD	Word - a word in a PDF file. Each word contains a sequence of characters in one or more styles (see Style).
HIG_PDF_WORDFINDER	WordFinder - extracts words from a PDF file, and enumerates the words on a single page or on all pages in a document.

1.3.3.4.2.1 HIG_PDF_ACTION

Handle to a PDF action object, which is a task that is performed when a user clicks on a link or a bookmark. Action types include:

- Going to another view within the same document
- Going to a specified view in another PDF file
- Launching an arbitrary file
- Resolving a URL

See Section 8.5 in the PDF Reference for more information on actions.

Members:

IG_PDF_action_create	Creates a new action object.
IG_PDF_action_create_destination	Creates a new action that takes the user to the specified destination view.
IG_PDF_action_create_filename	Creates an action of the specified type from a file name.
IG_PDF_action_delete	Deletes an action object.
IG_PDF_action_get_destination	Gets an action's destination view.
IG_PDF_action_get_dictionary	Gets the dictionary corresponding to an action.
IG_PDF_action_get_filename	Gets a file name from an action.
IG_PDF_action_get_type	Gets an action's type.

1.3.3.4.2.1.1 IG_PDF_action_create

Creates a new action object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_create(
        HIG_PDF_DOC hDoc,
        HIG_PDF_ATOM hType,
        LPHIG_PDF_ACTION lphAction
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the action is created and used.
hType	HIG_PDF_ATOM	The atom corresponding to the action's subtype.
lphAction	LPHIG_PDF_ACTION	The newly created action object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.1.2 IG_PDF_action_create_destination

Creates a new action that takes the user to the specified destination view.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_create_destination(
        HIG_PDF_DOC hDoc,
        HIG_PDF_DESTINATION hDest,
        LPHIG_PDF_ACTION lphAction
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the action is created and used.
hDest	HIG_PDF_DESTINATION	The destination.
lphAction	LPHIG_PDF_ACTION	The newly created action object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function can only be used for destinations in the same document as the source document. Cross-document links must be built up from the base level, populating the Action dictionary for the GotoR action as described in Section 8.5.3 in the PDF Reference.

1.3.3.4.2.1.3 IG_PDF_action_create_filename

Creates an action of the specified type from a file name.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_create_filename(
        HIG_PDF_DOC hDoc,
        HIG_PDF_ATOM hType,
        LPCSTR szFileName,
        LPHIG_PDF_ACTION lphAction
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the action is created and used.
hType	HIG_PDF_ATOM	The type of action to create.
szFileName	LPCSTR	The file name.
lphAction	LPHIG_PDF_ACTION	The newly created action object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.1.4 IG_PDF_action_delete

Deletes an action object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_delete(
        HIG_PDF_ACTION hAction
);
```

Arguments:

Name	Type	Description
hAction	HIG_PDF_ACTION	The action object to delete.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.1.5 IG_PDF_action_get_destination

Gets an action's destination view.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_get_destination(
        HIG_PDF_ACTION hAction,
        LPHIG_PDF_DESTINATION lphDest
);
```

Arguments:

hAction	HIG_PDF_ACTION	The action whose destination is obtained.
lphDest	HIG_PDF_DESTINATION	The action's destination, which may be either an explicit or named (basic string or name object). Use the IG_PDF_destination_resolve on this returned value to obtain an explicit destination.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This only works for actions that contain a view destination - that is, actions whose type is GoTo. For named destinations, this function may return a basic string or name object. See Section 8.2.1 in the PDF Reference for more information on named destinations.

Since this function may not return an explicit destination, use the IG_PDF_destination_resolve on the returned value to obtain an explicit destination.

1.3.3.4.2.1.6 IG_PDF_action_get_dictionary

Gets the dictionary corresponding to an action.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_get_dictionary(
        HIG_PDF_ACTION hAction,
        LPHIG_PDF_BASOBJ lphDictionary
);
```

Arguments:

Name	Type	Description
hAction	HIG_PDF_ACTION	The action whose dictionary is obtained.
lphDictionary	LPHIG_PDF_BASOBJ	Dictionary object for the action.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.1.7 IG_PDF_action_get_filename

Gets a file name from an action.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_get_filename(
        HIG_PDF_ACTION hAction,
        LPSTR lpBuf,
        AT_INT nSize,
        LPAT_INT lpnLen
);
```

Arguments:

Name	Type	Description
hAction	HIG_PDF_ACTION	The action whose file name is obtained.
lpBuf	LPSTR	The buffer to return a file name.
nSize	AT_INT	Size of lpBuf.
lpnLen	LPAT_INT	Length of the file name.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Not all types of actions have file names; this function only works for actions that contain a file specification. See Section 8.5 in the PDF Reference for more information on the contents of various types of actions.

1.3.3.4.2.1.8 IG_PDF_action_get_type

Gets an action's type.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_action_get_type(
        HIG_PDF_ACTION hAction,
        LPHIG_PDF_ATOM lphType
);
```

Arguments:

Name	Type	Description
hAction	HIG_PDF_ACTION	The action whose type is obtained.
lphType	LPHIG_PDF_ATOM	The atom corresponding to the action's type.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.2 HIG_PDF_ATOM

Handle to the PDF atom object. A hashed token used in place of strings to optimize performance (it is much faster to compare Atoms than strings). Many methods use Atoms.

Members:

IG_PDF_atom_from_string	Gets the Atom for the specified string.
IG_PDF_atom_get_string	Gets the string associated with the specified Atom.

1.3.3.4.2.2.1 IG_PDF_atom_from_string

Gets the Atom for the specified string.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_atom_from_string(
        LPCSTR lpString,
        LPHIG_PDF_ATOM lphAtom
);
```

Arguments:

Name	Type	Description
lpString	LPCSTR	The string for which an atom is obtained.
lphAtom	LPHIG_PDF_ATOM	Atom return value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

You can also use this function to create an Atom, since it creates one for the string if one does not already exist.

If an Atom already exists for lpString, the existing Atom is returned. Thus Atoms may be compared for equality of the underlying string.

Because Atoms cannot be deleted, they are useful for strings that are used many times, but are not advisable for strings that have a short lifetime. For the same reason, it is not a good idea to create large numbers of Atoms.

1.3.3.4.2.2.2 IG_PDF_atom_get_string

Gets the string associated with the specified Atom.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_atom_get_string(
        HIG_PDF_ATOM hAtom,
        LPCSTR* lpString
);
```

Arguments:

Name	Type	Description
hAtom	HIG_PDF_ATOM	The Atom whose string is obtained.
lpString	LPCSTR*	The string corresponding to hAtom. Returns an empty string if hAtom is equal to IG_PDF_ATOM_NULL or NULL if the hAtom has not been defined.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3 HIG_PDF_BOOKMARK

Handle to a PDF bookmark object, which allows the user to navigate interactively from one part of the document to another. It consists of a tree-structured hierarchy of bookmarks. Each bookmark has:

- A title that appears on screen
- An action that specifies what happens when the user clicks on the bookmark

The typical action for a user-created bookmark is to move to another location in the current document or outside of it. Each bookmark in the bookmark tree structure has zero or more children that appear indented on screen, and zero or more siblings that appear at the same indentation level. All bookmarks except the bookmark at the top level of the hierarchy have an one parent, i.e., the bookmark under which it is indented. A bookmark is said to be open if its children are visible on the screen, and closed if they are not.

See the section 8.2.2, "Document Outline," in the PDF Reference for more information on bookmarks.

Members:

IG_PDF_bookmark_add_child	Adds hChildBookmark as the last child of parent.
IG_PDF_bookmark_add_new_child	Adds a new bookmark to the tree containing hBookmark.
IG_PDF_bookmark_add_next	Adds hNewNext as the new right sibling to hBookmark.
IG_PDF_bookmark_add_new_sibling	Adds a new bookmark to the tree containing hBookmark as the new right sibling.
IG_PDF_bookmark_add_prev	Adds hNewPrev as the new left sibling to hBookmark.
IG_PDF_bookmark_add_subtree	Adds a copy of the bookmark sub-tree source to hBookmark.
IG_PDF_bookmark_delete	Deletes a bookmark object.
IG_PDF_bookmark_find_title	Gets the first bookmark whose title is lpTitle.
IG_PDF_bookmark_get_action	Gets hBookmark's action.
IG_PDF_bookmark_get_color	Gets the color of the specified bookmark.
IG_PDF_bookmark_get_count	Gets the number of open bookmarks in a sub-tree.
IG_PDF_bookmark_get_first_child	Gets hBookmark's first child.
IG_PDF_bookmark_get_flags	Gets the flags of the specified bookmark.
IG_PDF_bookmark_get_indent	Gets the indentation level of a bookmark in its containing tree.
IG_PDF_bookmark_get_last_child	Gets hBookmark's last child.
IG_PDF_bookmark_get_next	Gets hBookmark's next (right) sibling.
IG_PDF_bookmark_get_parent	Gets hBookmark's parent bookmark.
IG_PDF_bookmark_get_prev	Gets hBookmark's previous (left) sibling.
IG_PDF_bookmark_get_title	Gets hBookmark's title.
IG_PDF_bookmark_has_children	Tests whether a bookmark has children.
IG_PDF_bookmark_is_open	Tests whether a bookmark is open.
IG_PDF_bookmark_remove	Removes hBookmark sub-tree from the bookmark tree containing it.
IG_PDF_bookmark_remove_action	Removes hBookmark's action.
IG_PDF_bookmark_set_action	Sets hBookmark's action.
IG_PDF_bookmark_set_color	Sets hBookmark's color.
IG_PDF_bookmark_set_flags	Sets the flags of the specified bookmark.
IG_PDF_bookmark_set_open	Opens or closes a bookmark.
IG_PDF_bookmark_set_title	Sets hBookmark's title.
IG_PDF_bookmark_unlink	Unlinks a bookmark from the bookmark tree that contains it.

1.3.3.4.2.3.1 IG_PDF_bookmark_add_child

Adds hChildBookmark as the last child of parent, adjusting the tree containing parent appropriately.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_child(
        HIG_PDF_BOOKMARK hBookmark,
        HIG_PDF_BOOKMARK hChildBookmark
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The parent of the bookmark being added.
hChildBookmark	HIG_PDF_BOOKMARK	The bookmark that will become the last child of hBookmark. hChildBookmark must have been previously unlinked.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If parent previously had no children, it is open after the child is added.

1.3.3.4.2.3.2 IG_PDF_bookmark_add_new_child

Adds a new bookmark to the tree containing hBookmark as the new last child of hBookmark.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_new_child(
        HIG_PDF_BOOKMARK hBookmark,
        LPSTR lpszInitialText,
        LPHIG_PDF_BOOKMARK lphChildBookmark
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark to which a new last child is added.
lpszInitialText	LPSTR	The new bookmark's title.
lphChildBookmark	LPHIG_PDF_BOOKMARK	The newly created bookmark.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If hBookmark previously had no children, it will be open after the child is added.

1.3.3.4.2.3.3 IG_PDF_bookmark_add_next

Adds hNewNext as the new right sibling to hBookmark.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_next(
        HIG_PDF_BOOKMARK hBookmark,
        HIG_PDF_BOOKMARK hNewNext
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark that will receive a new right sibling.
hNewNext	HIG_PDF_BOOKMARK	The bookmark to become the new right sibling of hBookmark. hNewNext must have been previously unlinked.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.4 IG_PDF_bookmark_add_new_sibling

Adds a new bookmark to the tree containing hBookmark as the new right sibling.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_new_sibling(
        HIG_PDF_BOOKMARK hBookmark,
        LPSTR lpszInitialText,
        LPHIG_PDF_BOOKMARK lphSiblingBookmark
);
```

Arguments:

Name	**Type**	**Description**
hBookmark	HIG_PDF_BOOKMARK	The bookmark that will be the left sibling of the new bookmark.
lpszInitialText	LPSTR	The new bookmark's title.
lphSiblingBookmark	LPHIG_PDF_BOOKMARK	The newly created bookmark.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.5 IG_PDF_bookmark_add_prev

Adds hNewPrev as the new left sibling to hBookmark, adjusting the tree containing hBookmark appropriately.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_prev(
        HIG_PDF_BOOKMARK hBookmark,
        HIG_PDF_BOOKMARK hNewPrev
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark that will receive a new left sibling.
hNewPrev	HIG_PDF_BOOKMARK	The bookmark to become the new left sibling of hBookmark. hNewPrev must have been previously unlinked.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.6 IG_PDF_bookmark_add_subtree

Adds a copy of the bookmark sub-tree source to hBookmark as a new last child of hBookmark.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_add_subtree(
        HIG_PDF_BOOKMARK hBookmark,
        HIG_PDF_BOOKMARK hSubtree,
        LPSTR lpszSourceTitle
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark to which the sub-tree source is added as a new last child.
hSubtree	HIG_PDF_BOOKMARK	The bookmark sub-tree to add.
lpszSourceTitle	LPSTR	The new bookmark's title.

Remarks:

This new item will have the text value lpszSourceTitle, will be open, and will have no destination attribute. hSubtree must have been previously unlinked. If hBookmark previously had no children, it will be open after the sub-tree is added.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.7 IG_PDF_bookmark_delete

Deletes a bookmark object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_delete(
        HIG_PDF_BOOKMARK hBookmark
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark object to delete.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.8 IG_PDF_bookmark_find_title

Gets the first bookmark whose title is lpTitle.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_find_title(
        HIG_PDF_BOOKMARK hBookmark,
        LPSTR lpTitle,
        AT_INT nTitleLen,
        AT_INT nMaxDepth,
        LPHIG_PDF_BOOKMARK lphBookmark
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The root of the bookmark sub-tree to search.
lpTitle	LPSTR	The text value for which to search.
nTitleLen	AT_INT	The length of lpTitle.
nMaxDepth	AT_INT	The number of sub-tree levels to search, not counting the root level. ● 0 - Only look at hBookmark, not at any of its children. ● 1 - Check hBookmark and its children, but not any grandchildren or great grandchildren, and so on. ● -1 - Check the entire sub-tree.
lphBookmark	LPHIG_PDF_BOOKMARK	The bookmark with the specified title or NULL if there is no such bookmark.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.9 IG_PDF_bookmark_get_action

This function gets hBookmark's action.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_action(
        HIG_PDF_BOOKMARK hBookmark,
        LPHIG_PDF_ACTION lphAction
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose action is obtained.
lphAction	LPHIG_PDF_ACTION	The bookmark's action.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.10 IG_PDF_bookmark_get_color

Gets the color of the specified bookmark.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_color(
        HIG_PDF_BOOKMARK hBookmark,
        LPAT_PDF_COLORVALUE lpBookmarkColor
);
```

Arguments:

hBookmark	HIG_PDF_BOOKMARK	The bookmark whose color is obtained.
lpBookmarkColor	LPAT_PDF_COLORVALUE	Color of the bookmark.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.11 IG_PDF_bookmark_get_count

Gets the number of open bookmarks in a sub-tree.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_count(
        HIG_PDF_BOOKMARK hBookmark,
        LPAT_INT lpnCount
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The root bookmark of a sub-tree to count.
lpnCount	LPAT_INT	Number of open bookmarks in the sub-tree (not including hBookmark).

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.12 IG_PDF_bookmark_get_first_child

Gets hBookmark's first child.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_first_child(
        HIG_PDF_BOOKMARK hBookmark,
        LPHIG_PDF_BOOKMARK lphFirstChild
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose first child is obtained.
lphFirstChild	LPHIG_PDF_BOOKMARK	First child of hBookmark or NULL, if hBookmark has no children.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.13 IG_PDF_bookmark_get_flags

Gets the flags of the specified bookmark.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_flags(
        HIG_PDF_BOOKMARK hBookmark,
        LPAT_INT lpnFlags
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose flags are obtained.
lpnFlags	LPAT_INT	Bookmark's flags. The OR value of the enumIGPDFBookmarkFlags.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.14 IG_PDF_bookmark_get_indent

Gets the indentation level of a bookmark in its containing tree.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_indent(
        HIG_PDF_BOOKMARK hBookmark,
        LPAT_INT lpnIndent
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose indentation level is obtained.
lpnIndent	LPAT_INT	The indentation level of hBookmark in its containing tree. The root level has an indentation level of zero.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.15 IG_PDF_bookmark_get_last_child

Gets hBookmark's last child.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_last_child(
        HIG_PDF_BOOKMARK hBookmark,
        LPHIG_PDF_BOOKMARK lphLastChild
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose last child is obtained.
lphLastChild	LPHIG_PDF_BOOKMARK	Last child of hBookmark or NULL if hBookmark has no children.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.16 IG_PDF_bookmark_get_next

Gets hBookmark's next (right) sibling.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_next(
        HIG_PDF_BOOKMARK hBookmark,
        LPHIG_PDF_BOOKMARK lphNext
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose right sibling is obtained.
lphNext	LPHIG_PDF_BOOKMARK	hBookmark's next (right) sibling or NULL if hBookmark has no next sibling (it is its parent's last child).

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.17 IG_PDF_bookmark_get_parent

Gets hBookmark's parent bookmark.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_parent(
        HIG_PDF_BOOKMARK hBookmark,
        LPHIG_PDF_BOOKMARK lphParent
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose parent is obtained.
lphParent	LPHIG_PDF_BOOKMARK	Parent bookmark of hBookmark or NULL, if hBookmark is the root of its tree.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.18 IG_PDF_bookmark_get_prev

Gets hBookmark's previous (left) sibling.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_prev(
        HIG_PDF_BOOKMARK hBookmark,
        LPHIG_PDF_BOOKMARK lphPrev
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose left sibling is obtained.
lphPrev	LPHIG_PDF_BOOKMARK	Previous (left) sibling of hBookmark or NULL, if hBookmark has no previous sibling (it is its parent's first child).

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.19 IG_PDF_bookmark_get_title

Gets hBookmark's title.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_get_title(
        HIG_PDF_BOOKMARK hBookmark,
        LPSTR szBuffer,
        AT_INT nSize,
        LPAT_INT lpnBytes
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose title is obtained.
szBuffer	LPSTR	Buffer into which the title will be written. If szBuffer is non-NULL, its length is assumed to be nSize + 1, because a null byte is appended to the title.
nSize	AT_INT	The size of szBuffer.
lpnBytes	LPAT_INT	The number of bytes copied into szBuffer, not counting the trailing null byte. If szBuffer is NULL, the number of bytes in the bookmark is returned.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.20 IG_PDF_bookmark_has_children

This function tests whether a bookmark has children or not.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_has_children(
        HIG_PDF_BOOKMARK hBookmark,
        LPAT_PDF_BOOL lpbHasChildren
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark to test.
lpbHasChildren	LPAT_PDF_BOOL	TRUE if hBookmark has any children; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.21 IG_PDF_bookmark_is_open

Tests whether a bookmark is open.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_is_open(
        HIG_PDF_BOOKMARK hBookmark,
        LPAT_PDF_BOOL lpbOpen
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark to test.
lpbOpen	LPAT_PDF_BOOL	TRUE if hBookmark is open; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

An open bookmark shows all its children.

1.3.3.4.2.3.22 IG_PDF_bookmark_remove

Removes hBookmark sub-tree from the bookmark tree containing it.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_remove(
        HIG_PDF_BOOKMARK hBookmark
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The root bookmark of the sub-tree to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.23 IG_PDF_bookmark_remove_action

Removes hBookmark's action.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_remove_action(
        HIG_PDF_BOOKMARK hBookmark
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose action is removed.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.24 IG_PDF_bookmark_set_action

Sets hBookmark's action.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_action(
        HIG_PDF_BOOKMARK hBookmark,
        HIG_PDF_ACTION hAction
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose action is set.
hAction	HIG_PDF_ACTION	The bookmark's action to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.25 IG_PDF_bookmark_set_color

Sets hBookmark's color.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_color(
        HIG_PDF_BOOKMARK hBookmark,
        LPAT_PDF_COLORVALUE lpBookmarkColor
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose color is set.
lpBookmarkColor	LPAT_PDF_COLORVALUE	The bookmark's color to set. Must be in IG_PDF_DEVICE_RGB.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.26 IG_PDF_bookmark_set_flags

Sets the flags of the specified bookmark.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_flags(
        HIG_PDF_BOOKMARK hBookmark,
        AT_INT nFlags
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose flags are set.
nFlags	AT_INT	Bookmark's flags. The OR value of the enumIGPDFBookmarkFlags.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.27 IG_PDF_bookmark_set_open

Opens or closes a bookmark.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_open(
        HIG_PDF_BOOKMARK hBookmark,
        AT_PDF_BOOL bIsOpen
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark to open or close.
bIsOpen	AT_PDF_BOOL	TRUE if the bookmark is opened; FALSE if the bookmark is closed.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

An open bookmark shows its children, while a closed bookmark does not.

1.3.3.4.2.3.28 IG_PDF_bookmark_set_title

This function sets hBookmark's title.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_set_title(
        HIG_PDF_BOOKMARK hBookmark,
        LPCSTR lpTitle,
        AT_INT nTitleLen
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark whose title is set.
lpTitle	LPCSTR	String containing the bookmark's new title.
nTitleLen	AT_INT	The size of lpTitle.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.3.29 IG_PDF_bookmark_unlink

Unlinks a bookmark from the bookmark tree that contains it, and adjusts the tree appropriately.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_bookmark_unlink(
        HIG_PDF_BOOKMARK hBookmark
);
```

Arguments:

Name	Type	Description
hBookmark	HIG_PDF_BOOKMARK	The bookmark to unlink.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.4 HIG_PDF_DESTINATION

Handle to a PDF destination object, which represents a particular view of a page in a document. It contains a reference to a page, a rectangle on that page, and information specifying how to adjust the view to fit the window's size and shape. See section 8.2, "Document-Level Navigation," in the PDF Reference for more information on destinations.

Members:

IG_PDF_destination_create	Creates a new destination object.
IG_PDF_destination_delete	Deletes a destination object.
IG_PDF_destination_get_explicit_attrs	Gets a destination's fit type, destination rectangle, and zoom factor.
IG_PDF_destination_get_named_attrs	Gets a destination's named attributes.
IG_PDF_destination_get_type	Gets a destination's type.
IG_PDF_destination_remove	Removes a view destination object.
IG_PDF_destination_resolve	Resolves a destination.

1.3.3.4.2.4.1 IG_PDF_destination_create

Creates a new destination object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_create(
        HIG_PDF_DOC hDoc,
        HIG_PDF_PAGE hPage,
        HIG_PDF_ATOM hInitialFitType,
        LPAT_PDF_FIXEDRECT lpInitialRect,
        AT_PDF_FIXED nInitialZoom,
        LPHIG_PDF_DESTINATION lphDest
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the destination is used.
hPage	HIG_PDF_PAGE	The destination page.
hInitialFitType	HIG_PDF_ATOM	Destination fit type. Must be one of the View Destination Fit Types.
lpInitialRect	LPAT_PDF_FIXEDRECT	Pointer to a AT_PDF_FIXEDRECT specifying the destination rectangle, specified in user space coordinates. The appropriate information will be extracted from lpInitialRect, depending on hInitialFitType, to create the destination. All four of lpInitialRect's components should be set.
nInitialZoom	AT_PDF_FIXED	The zoom factor to set for the destination. Used only if hInitialFitType is XYZ. Use the predefined value IG_PDF_DEST_NULL to indicate a NULL zoom factor.
lphDest	LPHIG_PDF_DESTINATION	The newly created destination object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.4.2 IG_PDF_destination_delete

Deletes a destination object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_delete(
        HIG_PDF_DESTINATION hDest
);
```

Arguments:

Name	Type	Description
hDest	HIG_PDF_DESTINATION	The destination object to delete.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.4.3 IG_PDF_destination_get_explicit_attrs

Gets a destination's fit type, destination rectangle, and zoom factor.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_get_explicit_attrs(
        HIG_PDF_DESTINATION hDest,
        LPAT_INT lpnPageNum,
        LPHIG_PDF_ATOM lphFitType,
        LPAT_PDF_FIXEDRECT lpRect,
        LPAT_PDF_FIXED lpnZoom
);
```

Arguments:

Name	Type	Description
hDest	HIG_PDF_DESTINATION	The destination whose explicit attributes are obtained.
lpnPageNum	LPAT_INT	The page number of the destination's page.
lphFitType	LPHIG_PDF_ATOM	Destination fit type. One of the Destination Fit Types values.
lpRect	AT_PDF_FIXEDRECT	Pointer to a AT_PDF_FIXEDRECT containing the destination's rectangle, specified in user space coordinates.
lpnZoom	LPAT_PDF_FIXED	The destination's zoom factor.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Applies only to IG_PDF_DEST_EXPLICIT type of destination.

1.3.3.4.2.4.4 IG_PDF_destination_get_named_attrs

Gets a destination's named attributes.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_get_named_attrs(
        HIG_PDF_DESTINATION hDest,
        LPHIG_PDF_BASOBJ lphName
);
```

Arguments:

Name	Type	Description
hDest	HIG_PDF_DESTINATION	The destination whose named attributes are obtained.
lphName	LPHIG_PDF_BASOBJ	Basic array object for the destination. Returns NULL if the destination is invalid.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Applies only to IG_PDF_DEST_NAMED type of destination.

1.3.3.4.2.4.5 IG_PDF_destination_get_type

Gets a destination's type.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_get_type(
        HIG_PDF_DESTINATION hDest,
        LPAT_INT lpnType
);
```

Arguments:

Name	Type	Description
hDest	HIG_PDF_DESTINATION	The destination whose type is obtained.
lpnType	LPAT_INT	The destination type. One of the enumIGPDFDestinationType values.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.4.6 IG_PDF_destination_remove

Removes a view destination object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_remove(
        HIG_PDF_DESTINATION hDest
);
```

Arguments:

Name	Type	Description
hDest	HIG_PDF_DESTINATION	The destination to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.4.7 IG_PDF_destination_resolve

Resolves a destination.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_destination_resolve(
        HIG_PDF_DESTINATION hDest,
        HIG_PDF_DOC hDoc,
        LPHIG_PDF_DESTINATION lphResolvedDest
);
```

Arguments:

Name	Type	Description
hDest	HIG_PDF_DESTINATION	The destination whose type is obtained.
hDoc	HIG_PDF_DOC	The PDF document that contains the destination.
lphResolvedDest	LPHIG_PDF_DESTINATION	The resolved view destination.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

hDest is the value of the D key in an action. It can be a real destination (an array) or a name. If it is a name, look it up in hDoc's Dests dictionary. The value found there can be a real destination (an array) or a dictionary. If it's a dictionary, look up the D key in that dictionary. This function is useful for getting an explicit view destination from an action.

1.3.3.4.2.5 HIG_PDF_DICTIONARY

Handle to a PDF dictionary object, which represents an optional-content membership dictionary object.

Members:

IG_PDF_dictionary_create	Creates a new optional-content membership dictionary object in the given document for the given layers and visibility policy.
IG_PDF_dictionary_get_layer	Gets the layer with the specified index in a membership dictionary.
IG_PDF_dictionary_get_layer_count	Gets the number of layers listed in a membership dictionary.
IG_PDF_dictionary_get_unique_id	Returns some 32-bit integer that is unique for all Dictionary objects.
IG_PDF_dictionary_get_vis_policy	Gets the optional-content membership dictionary's visibility policy.
IG_PDF_dictionary_release	Releases the native object and frees memory.

1.3.3.4.2.5.1 IG_PDF_dictionary_create

Creates a new optional-content membership dictionary object in the given document for the given layers and visibility policy.

Declaration:

```
IG_PDF_dictionary_create(
        HIG_PDF_DOC hDoc,
        LPHIG_PDF_LAYER hLayers,
        AT_INT nLayersCount,
        AT_INT policy,
        LPHIG_PDF_DICTIONARY lphDictionary
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the dictionary is used.
hLayers	LPHIG_PDF_LAYER	Array of layers to be the members of the dictionary.
nLayersCount	AT_INT	The number of layers.
policy	AT_INT	The visibility policy that determines the visibility of content with respect to the ON/OFF state of the layers listed in the dictionary.
lphDictionary	LPHIG_PDF_DICTIONARY	The newly created dictionary object.

Return Value:

The newly created dictionary object, or NULL if no layers are supplied.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

To add layer to the existing dictionary, get the current layers' list, modify it, and then create a new dictionary with the new list of layers.

1.3.3.4.2.5.2 IG_PDF_dictionary_get_layer

Gets the layer with the specified index in a membership dictionary.

Declaration:

```
IG_PDF_dictionary_get_layer(
        HIG_PDF_DICTIONARY hDictionary,
        UINT nIndex,
        LPHIG_PDF_LAYER lpLayer
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_DICTIONARY	The membership dictionary whose layer is obtained.
nIndex	UINT	The index of the needed layer in the dictionary.
lpLayer	LPHIG_PDF_LAYER	The layer object.

Return Value:

Layer object.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.5.3 IG_PDF_dictionary_get_layer_count

Gets the number of layers listed in a membership dictionary.

Declaration:

```
IG_PDF_dictionary_get_layer_count(
        HIG_PDF_DICTIONARY hDictionary,
        LPUINT lpnCount
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_DICTIONARY	The membership dictionary whose layers count is obtained.
lpnCount	LPUINT	The count of the document's layers.

Return Value:

The count of the document's layers.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.5.4 IG_PDF_dictionary_get_unique_id

Returns some 32-bit integer that is unique for all Dictionary objects.

Declaration:

```
IG_PDF_dictionary_get_unique_id(
        HIG_PDF_DICTIONARY hDictionary,
        LPUINT lpnUniqueId
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_DICTIONARY	Dictionary object.
lpnUniqueId	LPUINT	The unique identifier.

Return Value:

An unique identifier of this Dictionary Object.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

It is guaranteed that there cannot be two Dictionary objects with the same UniqueIds.

Can be used for Dictionary objects' identification.

1.3.3.4.2.5.5 IG_PDF_dictionary_get_vis_policy

Gets the optional-content membership dictionary's visibility policy, which determines the visibility of content with respect to the ON-OFF state of the layers listed in the dictionary.

Declaration:

```
IG_PDF_dictionary_get_vis_policy(
        HIG_PDF_DICTIONARY hDictionary,
        LPAT_INT lpPolicy
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_DICTIONARY	The dictionary whose policy is obtained.
lpPolicy	LPAT_INT	The visibility policy.

Return Value:

The visibility policy.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.5.6 IG_PDF_dictionary_release

Releases the native object and frees memory.

Declaration:

```
IG_PDF_dictionary_release(
        HIG_PDF_DICTIONARY hDictionary
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_DICTIONARY	Dictionary object to release.

Return Value:

Nothing.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.6 HIG_PDF_DOC

Handle to the PDF document object. The underlying PDF representation of a document. Through PDF Document, your application can perform most of the Edit Pages operations (delete, replace, and so on). Thumbnails can be created and deleted through this object. You can set and retrieve document information fields through this object as well.

Members:

IG_PDF_doc_create	Creates a PDF document and attaches it to HMIGEAR.
IG_PDF_doc_create_new_page	Creates a new PDF page for the hMPIDoc.
IG_PDF_doc_delete_pages	Deletes the specified pages.
IG_PDF_doc_insert_pages	Inserts nPageCount pages from hDoc2 into hDoc.
IG_PDF_doc_get_bookmark	Gets the root of the document's bookmark tree.
IG_PDF_doc_get_info	Gets the value of a key in a document's Info dictionary, or the value of this same key in the XMP metadata, whichever is latest.
IG_PDF_doc_get_layer	Gets the layer with a specified index.
IG_PDF_doc_get_layer_count	Gets the layer count for the document.
IG_PDF_doc_get_page	Gets a handle to a specific page.
IG_PDF_doc_get_page_count	Gets the number of pages in the document.
IG_PDF_doc_get_root	Returns the Catalog dictionary of the PDF document
IG_PDF_doc_set_info	Sets the value of a key in a document's Info dictionary.
IG_PDF_doc_print	Prints PDF pages from a PDF document, allowing the user to specify options such as page size, rotation, and fit mode.
IG_PDF_doc_create_wordfinder	Creates a word finder that is used to extract text in the host encoding from a PDF file.
IG_PDF_doc_create_wordfinder_ucs	Creates a word finder that is used to extract text in the host encoding from a PDF file.
IG_PDF_doc_get_new_crypt_handler	Gets the specified document's new security handler (that is, the security handler that will be used after the document is saved).
IG_PDF_doc_get_new_security_data	Gets the security data structure for the specified document's new security handler.
IG_PDF_doc_get_new_security_info	Gets the security information from the specified document's new security handler.
IG_PDF_doc_get_security_data	Gets the security data structure for the specified document's new security handler.
IG_PDF_doc_page_release	Releases a handle to a PDF page.
IG_PDF_doc_perm_request	Checks the permissions associated with the specified document using the latest permissions format, and determines whether the requested operation is allowed for the specified object in the document.
IG_PDF_doc_set_new_crypt_handler	Sets specified document's new security handler (the security handler that will be used after the document is saved).
IG_PDF_doc_set_new_security_data	Sets the security data structure for the specified document's new security handler.

1.3.3.4.2.6.1 IG_PDF_doc_create

This function creates a PDF document and attaches it to HMIGEAR.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_create(
        HMIGEAR hMPIDoc
);
```

Arguments:

Name	Type	Description
hMPIDoc	HMIGEAR	ImageGear document to which to attach a PDF document.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function does not do anything if hMPIDoc is already vector document.

To obtain a handle to the PDF document, use the following:

```
HIG_PDF_DOC hPDFDoc = (HIG_PDF_DOC)NULL;
       IG_mpi_info_get( hMPIDoc, IG_MP_DOCUMENT, &hPDFDoc, sizeof(hPDFDoc) );
```

1.3.3.4.2.6.2 IG_PDF_doc_create_new_page

This function creates a new PDF page for the hMPIDoc.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_create_new_page(
        HMIGEAR hMPIDoc,
        LONG nAfterPage,
        LPAT_PDF_FIXEDRECT lpMediaBox
);
```

Arguments:

Name	Type	Description
hMPIDoc	HMIGEAR	The document in which the page is created.
nAfterPage	LONG	The page number after which the new page is inserted. The first page is 0. Use IG_PDF_BEFORE_FIRST_PAGE to insert the new page at the beginning of a document.
lpMediaBox	LPAT_PDF_FIXEDRECT	Rectangle specifying the page's media box, specified in user space coordinates.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The new PDF page is created at the specified position.

The previous value (if any) is not deleted with the IG_image_delete function. The size of the multi-page image is not changed, so that page arrays is not expanded when nAfterPage is greater than pageCount-1.

To obtain a handle to the PDF page, use the following:

```
HIGEAR hNewPage = NULL;
IG_mpi_page_get(m_hMPDoc, nAfterPage+1, &hNewPage);
HIG_PDF_PAGE hNewPDFPage = NULL;
IG_vector_data_get( hNewPage, (LPVOID*)&hNewPDFPage );
```

1.3.3.4.2.6.3 IG_PDF_doc_create_wordfinder

Creates a word finder that is used to extract text in the host encoding from a PDF file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_create_wordfinder(
        HIG_PDF_DOC hDoc,
        LPWORD lpOutEncInfo,
        LPCHAR* lpOutEncVec,
        LPCHAR* lpLigatureTbl,
        SHORT nAlgVersion,
        WORD nFlags,
        LPVOID lpClientData,
        LPHIG_PDF_WORDFINDER lphWordFinder
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document on which the word finder is used.
lpOutEncInfo	LPWORD	Array of 256 flags, specifying the type of character at each position in the encoding. Each flag is an OR of the Character Type Codes. If lpOutEncInfo is NULL, the platform's default encoding info is used. Use lpOutEncInfo and lpOutEncVec together; for every lpOutEncInfo use a corresponding lpOutEncVec to specify the character at that position in the encoding.
lpOutEncVec	LPCHAR*	Array of 256 null-terminated strings that are the glyph names in encoding order. See the discussion of character names in Section 5.3 of the PostScript Language Reference Manual, Third Edition. If lpOutEncVec is NULL, the platform's default encoding vector is used. Use this parameter with lpOutEncInfo.
lpLigatureTbl	LPCHAR*	A null-terminated array of null-terminated strings. Each string is the glyph name of a ligature in the font. When a word contains a ligature, the glyph name of the ligature is substituted for the ligature (for example, ff is substituted for the ff ligature). If ligatureTbl is NULL, a default ligature table is used, containing the following ligatures: fi, ff, fl, ffi, ffl, ch, cl, ct, ll, ss, fs, st, oe, OE.
nAlgVersion	SHORT	The version of the word-finding algorithm to use.
nFlags	WORD	Word-finding options that determine the tables filled when using IG_PDF_wordfinder_acquire_word_list. Must be an OR of one or more of enumIGPDFWordFlags.
lpClientData	LPVOID	Pointer to user-supplied data to pass to the newly created word finder. Set to NULL.
lphWordFinder	LPHIG_PDF_WORDFINDER	Handle to the new WordFinder.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The word finder also extracts text from Form XObjects that are executed in the page contents. For information about Form XObjects, see Section 4.9 in the PDF Reference.

This function also works for non-Roman (CJK or Chinese-Japanese-Korean) viewers. In this case, words are extracted to the host encoding. Users desiring Unicode output must use IG_PDF_doc_create_wordfinder_ucs, which does the extraction for Roman or non-Roman text.

The type of WordFinder determines the encoding of the string returned by IG_PDF_word_get_string. For instance, if IG_PDF_doc_create_wordfinder_ucs is used to create the word finder, IG_PDF_word_get_string returns only Unicode.

For CJK viewers, words are stored internally using CID encoding. For more information on CIDFonts and related topics, see Section 5.6 in the PDF Reference. For detailed information on CIDFonts, see Technical Note #5092, CID-Keyed Font Technology Overview, and Technical Note #5014, Adobe CMap and CIDFont Files Specification.

1.3.3.4.2.6.4 IG_PDF_doc_create_wordfinder_ucs

Creates a word finder that is used to extract text in the host encoding from a PDF file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_create_wordfinder_ucs(
        HIG_PDF_DOC hDoc,
        SHORT nAlgVersion,
        WORD nFlags,
        LPVOID lpClientData,
        LPHIG_PDF_WORDFINDER lphWordFinder
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document on which the word finder is used.
nAlgVersion	SHORT	The version of the word-finding algorithm to use.
nFlags	WORD	Word-finding options that determine the tables filled when using IG_PDF_wordfinder_acquire_word_list. Must be an OR of one or more of enumIGPDFWordFlags.
lpClientData	LPVOID	Pointer to user-supplied data to pass to the newly created word finder. Set to NULL.
lphWordFinder	LPHIG_PDF_WORDFINDER	Handle to the new WordFinder.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The word finder also extracts text from Form XObjects that are executed in the page contents. For information about Form XObjects, see Section 4.9 in the PDF Reference.

This function also works for non-Roman (CJK or Chinese-Japanese-Korean) viewers. In this case, words are extracted to the host encoding. Users desiring Unicode output must use IG_PDF_doc_create_wordfinder_ucs, which does the extraction for Roman or non-Roman text.

The type of WordFinder determines the encoding of the string returned by IG_PDF_word_get_string. For instance, if IG_PDF_doc_create_wordfinder_ucs is used to create the word finder, IG_PDF_word_get_string returns only Unicode.

For CJK viewers, words are stored internally using CID encoding. For more information on CIDFonts and related topics, see Section 5.6 in the PDF Reference. For detailed information on CIDFonts, see Technical Note #5092, CID-Keyed Font Technology Overview, and Technical Note #5014, Adobe CMap and CIDFont Files Specification.

1.3.3.4.2.6.5 IG_PDF_doc_delete_pages

Deletes the specified pages.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_delete_pages(
        HIG_PDF_DOC hDoc,
        LONG nStartPage,
        LONG nPageCount
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document from which pages are deleted.
nStartPage	LONG	The page number of the first page to delete. The first page is 0.
nPageCount	LONG	The number of pages to delete.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.6.6 IG_PDF_doc_get_bookmark

Gets the root of the document's bookmark tree.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_bookmark(
        HIG_PDF_DOC hDoc,
        LPHIG_PDF_BOOKMARK lphRootBookmark
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose root bookmark is obtained.
lphRootBookmark	LPHIG_PDF_BOOKMARK	The document's root bookmark.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The lphRootBookmark value is valid even if document's bookmark tree is empty.

1.3.3.4.2.6.7 IG_PDF_doc_get_info

This function can be used to obtain the values of the following standard document information dictionary keys: "Title", "Author", "Subject", "Keywords", "Creator", "Producer", "Created", and "Modified".

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_info(
        HIG_PDF_DOC hDoc,
        LPCSTR szInfoKey,
        LPSTR szBuffer,
        LONG nSize,
        LPLONG lpnBytes
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose Info dictionary key is obtained.
szInfoKey	LPCSTR	The name of the Info dictionary key whose value is obtained.
szBuffer	LPSTR	Result buffer containing the value associated with infoKey. If buffer is NULL, the method will just return the number of bytes required.
nSize	LONG	The maximum number of bytes that can be written into buffer.
lpnBytes	LPLONG	If szBuffer is NULL, the number of bytes in the specified key's value. If szBuffer is not NULL, returns the number of bytes copied into buffer, excluding the terminating NULL. You must pass at least the length + 1 as the buffer size since the routine adds a '\0' terminator to the data, even though the data is not a C string (it can contain embedded '\0's).

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

See Section 10.2.1 in the PDF Reference for information about Info dictionaries. All values in the Info dictionary should be strings; other data types such as numbers and Booleans should not be used as values in the Info dictionary.

Users may define their own Info dictionary entries. In this case, it is strongly recommended that the key have the developer's prefix assigned by the Adobe Solutions Network.

1.3.3.4.2.6.8 IG_PDF_doc_get_layer

Gets the layer with a specified index.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_layer(
        HIG_PDF_DOC hDoc,
        UINT nIndex,
        LPHIG_PDF_LAYER lphLayer
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	Document whose layer is obtained.
nIndex	UINT	Index of the layer to obtain.
lphLayer	LPHIG_PDF_LAYER	The obtained layer.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.6.9 IG_PDF_doc_get_layer_count

Gets the layer count for the document.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_layer_count(
        HIG_PDF_DOC hDoc,
        LPUINT lpnCount
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	Document whose layer count is obtained.
lpnCount	LPUINT	Layer count.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.6.10 IG_PDF_doc_get_new_crypt_handler

Gets the specified document's new security handler (that is, the security handler that will be used after the document is saved).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_new_crypt_handler(
        HIG_PDF_DOC hDoc,
        LPHIG_PDF_ATOM lphCryptHandler
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose new security handler is obtained.
lphCryptHandler	LPHIG_PDF_ATOM	The PDF atom corresponding to the name of the document's new security handler. Returns IG_PDF_ATOM_NULL if the document does not have a new security handler.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If the document does not have a new security handler, returns the document's current security handler.

1.3.3.4.2.6.11 IG_PDF_doc_get_new_security_data

Gets the security data structure for the specified document's new security handler.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_new_security_data(
        HIG_PDF_DOC hDoc,
        LPAT_PDF_SECURITYDATA* lppSecData
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose new security data structure is obtained.
lppSecData	LPAT_PDF_SECURITYDATA*	The security data structure for the document's new security handler.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Use IG_PDF_doc_get_security_data to get the security data structure for the document's current security handler.

The password strings in PDF are padded or truncated to exactly 32 bytes. If the password string is more than 32 bytes long, used only its first 32 bytes; if it is less than 32 bytes long, it padded by appending the required number of additional bytes from the beginning of the following padding string: < 28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >

1.3.3.4.2.6.12 IG_PDF_doc_get_new_security_info

Gets the security information from the specified document's new security handler.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_new_security_info(
        HIG_PDF_DOC hDoc,
        LPUINT lpnSecInfo
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose new security information is obtained.
lpnSecInfo	LPUINT	The document's new security information. The OR value of the enumIGPDFSecurityInfoFlags.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

No permissions are required to call this function.

1.3.3.4.2.6.13 IG_PDF_doc_get_page

This function obtains a handle to a PDF page.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_page(
        HIG_PDF_DOC hDoc,
        UINT nPageNumber,
        LPHIG_PDF_PAGE lphPage
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document from which the page is obtained.
nPageNumber	UINT	The index of the page to obtain. The first page is 0.
lphPage	LPHIG_PDF_PAGE	A pointer to memory that is populated with an HIG_PDF_PAGE handle for the selected page.

Return Value:

The number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Applications that obtain an HIG_PDF_PAGE from IG_PDF_doc_get_page are required to invoke this function to decrement that PDF page's reference count and release the HIG_PDF_PAGE instance. This ensures that the PDF document can successfully close.

For example:

```
const AT_INT FIRST_PAGE = 0 ;
HMIGEAR higDoc = 0 ;
HIG_PDF_DOC hPdfDoc = 0 ;
HIG_PDF_PAGE hPdfPage = 0 ;
UINT annotation_count = (UINT)-1 ;
// Recover number of annotations on first PDF page
IG_mpi_create( &higDoc , 0 ) ;
IG_mpi_file_open( "sample.pdf" , higDoc, IG_FORMAT_PDF , IG_MP_OPENMODE_READWRITE ) ;
IG_mpi_info_get( higDoc, IG_MP_DOCUMENT, &hPdfDoc, sizeof( hPdfDoc ) ) ;
IG_PDF_doc_get_page( hPdfDoc, FIRST_PAGE, &hPdfPage ) ;
IG_PDF_page_get_annotation_count( hPdfPage , &annotation_count ) ;
IG_PDF_doc_page_release( hPdfPage ) ;
IG_mpi_delete( higDoc ) ;
```

1.3.3.4.2.6.14 IG_PDF_doc_get_page_count

Gets the number of pages in the document.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_page_count(
        HIG_PDF_DOC hDoc,
        LPUINT lpnCount
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document for which the number of pages is obtained.
lpnCount	LPUINT	The number of pages in the document.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.6.15 IG_PDF_doc_get_root

This function returns the Catalog dictionary of the PDF document (this Catalog dictionary is the root of a PDF document`s object hierarchy).

Declaration:

```
AT_VOID IG_PDF_doc_get_root(
        HIG_PDF_DOC hDoc,
        LPHIG_PDF_BASOBJ lphRootDictionary
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The PDF document.
lphRootDictionary	LPHIG_PDF_BASOBJ	The resulting Catalog dictionary.

Return Value:

None

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

See also the PDF Reference, ch.3.6.

1.3.3.4.2.6.16 IG_PDF_doc_get_security_data

Gets the security data structure for the specified document's current security handler.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_get_security_data(
        HIG_PDF_DOC hDoc,
        LPAT_PDF_SECURITYDATA* lppSecData
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose security data structure is obtained.
lppSecData	LPAT_PDF_SECURITYDATA*	A pointer to the document's current security data structure.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Use IG_PDF_doc_get_new_security_data to get the data structure for the document's new security handler.

The password strings in PDF are padded or truncated to exactly 32 bytes. If the password string is more than 32 bytes long, used only its first 32 bytes; if it is less than 32 bytes long, it padded by appending the required number of additional bytes from the beginning of the following padding string: < 28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >

1.3.3.4.2.6.17 IG_PDF_doc_insert_pages

This function inserts nPageCount pages from hDoc2 into hDoc.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_insert_pages(
        HIG_PDF_DOC hDoc,
        LONG nAfterThisPage,
        HIG_PDF_DOC hDoc2,
        LONG nStartPage,
        LONG nPageCount,
        WORD nInsertFlags
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document into which pages are inserted.
nAfterThisPage	LONG	The page number in hDoc after which pages from hDoc2 are inserted. The first page is 0. If IG_PDF_BEFORE_FIRST_PAGE is used, the pages are inserted before the first page in hDoc . Use IG_PDF_LAST_PAGE to insert pages after the last page in hDoc.
hDoc2	HIG_PDF_DOC	The document containing the pages that are inserted into hDoc.
nStartPage	LONG	The page number of the first page in hDoc2 to insert into hDoc. The first page is 0.
nPageCount	LONG	The number of pages in hDoc2 to insert into hDoc. Use IG_PDF_ALL_PAGES to insert all pages from hDoc2 into hDoc.
nInsertFlags	WORD	Flags that determine what additional information is copied from hDoc2 into hDoc. An OR of enumIGPDFInsertFlags constants: ● IG_PDF_INSERT_BOOKMARKS - Inserts bookmarks as well as pages. The bookmark tree of hDoc2 is merged into the bookmark tree of hDoc by copying it as a new first-level sub-tree of hDoc's bookmark tree root of which it becomes the last child. If hDoc has no bookmark tree, it acquires one identical to the bookmark tree from hDoc2. ● IG_PDF_INSERT_THREADS - Inserts threads as well as pages. ● IG_PDF_INSERT_ALL - Inserts all pages, regardless of nStartPage and nPageCount, and document data.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
AT_CHAR*     first_filename      = "first.pdf" ;
UINT         first_page_count    = 0 ;
HMIGEAR      first_hmigear       = 0 ;
HIG_PDF_DOC  first_hig_pdf_doc   = 0 ;

AT_CHAR*     second_filename     = "second.pdf" ;
UINT         second_page_count   = 0 ;
HMIGEAR      second_hmigear      = 0 ;
```

```
HIG_PDF_DOC  second_hig_pdf_doc  = 0 ;
AT_CHAR*     combined_filename   = "combined.pdf" ;
UINT         combined_page_count = 0 ;
/* Open first PDF */
IG_mpi_create( &first_hmigear , 0 ) ;
IG_mpi_file_open( first_filename, first_hmigear, IG_FORMAT_UNKNOWN,
                    IG_MP_OPENMODE_READONLY ) ;
IG_PDF_doc_create( first_hmigear ) ;
IG_mpi_info_get( first_hmigear, IG_MP_DOCUMENT, &first_hig_pdf_doc,
                    sizeof( first_hig_pdf_doc ) ) ;
IG_mpi_page_count_get( first_hmigear, &first_page_count ) ;
/* Open second PDF */
IG_mpi_create( &second_hmigear , 0 ) ;
IG_mpi_file_open( second_filename , second_hmigear, IG_FORMAT_UNKNOWN,
                    IG_MP_OPENMODE_READONLY) ;
IG_PDF_doc_create( second_hmigear ) ;
IG_mpi_info_get( second_hmigear, IG_MP_DOCUMENT, &second_hig_pdf_doc,
                    sizeof(second_hig_pdf_doc) ) ;
IG_mpi_page_count_get( second_hmigear, &second_page_count ) ;
/* Insert pages */
IG_PDF_doc_insert_pages( second_hig_pdf_doc , IG_PDF_LAST_PAGE,
                    first_hig_pdf_doc , 0 , first_page_count ,
IG_PDF_INSERT_ALL ) ;
/* Save combined PDF document */
combined_page_count = first_page_count + second_page_count ;
IG_mpi_file_save( combined_filename , second_hmigear, 0 , 0,
combined_page_count ,
                    IG_FORMAT_PDF , IG_MPI_SAVE_OVERWRITE ) ;
IG_mpi_close( first_hmigear ) ;
IG_mpi_delete( first_hmigear ) ;
IG_mpi_close( second_hmigear ) ;
IG_mpi_delete( second_hmigear ) ;
```

Remarks:

All annotations, and anything else associated with the page (such as a thumbnail image) are copied from the hDoc2 pages to the new pages in hDoc. This function does not insert pages, if hDoc is equal to hDoc2. The nInsertFlags controls whether bookmarks and threads are inserted along with the specified pages.

1.3.3.4.2.6.18 IG_PDF_doc_page_release

This function decrements the reference count for a PDF page.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_page_release(
        HIG_PDF_PAGE hPage
);
```

Arguments:

Name	Type	Description
hPage	HIG_PDF_PAGE	The page to release.

Return Value:

The number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Applications that obtain an HIG_PDF_PAGE from IG_PDF_doc_get_page are required to invoke this function to decrement that PDF page's reference count and release the HIG_PDF_PAGE instance. This ensures that the PDF document can successfully close.

For example:

```
const AT_INT FIRST_PAGE = 0 ;

HMIGEAR higDoc = 0 ;
HIG_PDF_DOC hPdfDoc = 0 ;
HIG_PDF_PAGE hPdfPage = 0 ;
UINT annotation_count = (UINT)-1 ;

// Recover number of annotations on first PDF page
IG_mpi_create( &higDoc , 0 ) ;
IG_mpi_file_open( "sample.pdf" , higDoc, IG_FORMAT_PDF , IG_MP_OPENMODE_READWRITE ) ;
IG_mpi_info_get( higDoc, IG_MP_DOCUMENT, &hPdfDoc, sizeof( hPdfDoc ) ) ;

IG_PDF_doc_get_page( hPdfDoc, FIRST_PAGE, &hPdfPage ) ;
IG_PDF_page_get_annotation_count( hPdfPage , &annotation_count ) ;
IG_PDF_doc_page_release( hPdfPage ) ;

IG_mpi_delete( higDoc ) ;
```

Applications that obtain an HIG_PDF_PAGE from IG_vector_data_get should instead use IG_mpi_delete. The HMIGEAR retains ownership of HIG_PDF_PAGE and is responsible for releasing it.

For example:

```
const AT_INT FIRST_PAGE = 0 ;

HMIGEAR higDoc = 0 ;
HIGEAR higPage = 0 ;
HIG_PDF_PAGE hPdfPage = 0 ;
AT_ERRCOUNT errCount = 0 ;
```

```
UINT annotation_count = (UINT)-1 ;

// Recover number of annotations on the first PDF page
IG_mpi_create( &higDoc , 0 ) ;
IG_mpi_file_open( "sample.pdf" , higDoc, IG_FORMAT_PDF , IG_MP_OPENMODE_READWRITE ) ;
IG_mpi_page_get( higDoc , FIRST_PAGE , &higPage ) ;

IG_vector_data_get( higPage, ( LPVOID* )&hPdfPage ) ;
IG_PDF_page_get_annotation_count( hPdfPage , &annotation_count ) ;

IG_mpi_delete( higDoc ) ;
```

1.3.3.4.2.6.19 IG_PDF_doc_perm_request

Checks the permissions associated with the specified document using the latest permissions format, and determines whether the requested operation is allowed for the specified object in the document.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_perm_request(
        HIG_PDF_DOC hDoc,
        UINT nReqObj,
        UINT nReqOpr,
        LPVOID lpAuthData,
        LPSHORT lpnReqStatus
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose permissions are being requested.
nReqObj	UINT	The target object of the permissions request.
nReqOpr	UINT	The target operation of the permissions request.
lpAuthData	LPVOID	A pointer to an authorization data (password string).
lpnReqStatus	LPSHORT	One of enumIGPDFPermReqStatus request status constants.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.6.20 IG_PDF_doc_print

Prints PDF pages from a PDF document, allowing the user to specify options such as page size, rotation, and fit mode.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_print(
        HIG_PDF_DOC hDoc,
        LPAT_PDF_PRINTOPTIONS lpPrintOptions
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	Handle to the document to print.
lpPrintOptions	LPAT_PDF_PRINTOPTIONS	Parameters to control printing.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Please refer to the MFC PDF sample for a complete code example of using the IG_PDF_doc_print function.

1.3.3.4.2.6.21 IG_PDF_doc_set_info

This function can be used to set new values for the following standard document information dictionary keys: "Title", "Author", "Subject", "Keywords", and "Creator".

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_set_info(
       HIG_PDF_DOC hDoc,
       LPCSTR szInfoKey,
       LPSTR szBuffer,
       LONG nSize
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose Info dictionary key is set.
szInfoKey	LPCSTR	The name of the Info dictionary key whose value is set.
szBuffer	LPSTR	Buffer containing the value to associate with szInfoKey.
nSize	LONG	The number of bytes in buffer.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The following standard document information dictionary keys are read-only: "Producer", "Created", and "Modified"; the library will overwrite the values of these keys on document save.

See Section 10.2.1 on Info dictionaries in the PDF Reference for information about Info dictionaries. All values in the Info dictionary should be strings; other data types such as numbers and Booleans should not be used as values in the Info dictionary. If an info dictionary key is specified that is not currently in the info dictionary, it is added to the dictionary.

1.3.3.4.2.6.22 IG_PDF_doc_set_new_crypt_handler

Sets specified document's new security handler (the security handler that will be used after the document is saved).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_set_new_crypt_handler(
        HIG_PDF_DOC hDoc,
        HIG_PDF_ATOM hNewCryptHandler
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose new security handler is set.
hNewCryptHandler	HIG_PDF_ATOM	The PDF atom for the name of the new security handler to use for the document. Use IG_PDF_ATOM_NULL to remove security from the document.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function returns with no action if the new security handler is the same as the old one.

1.3.3.4.2.6.23 IG_PDF_doc_set_new_security_data

Sets the security data structure for the specified document's new security handler.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_doc_set_new_security_data(
        HIG_PDF_DOC hDoc,
        LPAT_PDF_SECURITYDATA lpSecData
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document whose new security data structure is set.
lppSecData	LPAT_PDF_SECURITYDATA	Pointer to the new security data structure to set for doc.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The new security handler must have been previously set using IG_PDF_doc_set_new_crypt_handler.

1.3.3.4.2.7 HIG_PDF_FONT

Handle to a PDF font object that is used to draw text on a page. A HIG_PDF_FONT has a number of attributes, including an array of widths and the character encoding.

Members:

IG_PDF_font_get_bbox	Gets a Type 3 font's bounding box.
IG_PDF_font_get_charset	Gets the font's character set.
IG_PDF_font_get_cid_systeminfo	Gets Registry and Ordering information for a CIDFont.
IG_PDF_font_get_cid_system_supplement	Gets the SystemSupplement number of a CIDFont.
IG_PDF_font_get_descendant	Gets a Type 0 font's descendant.
IG_PDF_font_get_encoding_index	Gets a font's encoding index.
IG_PDF_font_get_encoding_name	Gets a string representing a font's encoding.
IG_PDF_font_get_font_matrix	Gets a font's matrix.
IG_PDF_font_get_metrics	Gets a font's metrics.
IG_PDF_font_get_name	Gets the name of a font.
IG_PDF_font_get_subtype	Gets a font's subtype.
IG_PDF_font_get_widths	Gets the advance width of every glyph in a font.
IG_PDF_font_is_embedded	Tests whether the specified font is embedded in the PDF file.
IG_PDF_font_set_metrics	Sets a font's metrics.
IG_PDF_font_translate_string	Translates a string from the hFont's encoding into host encoding.
IG_PDF_font_translate_to_host	Translates a string from the hFont's encoding to host encoding.
IG_PDF_font_translate_to_ucs	Translates a string from whatever encoding the hFont uses to Unicode encoding.
IG_PDF_font_translate_widths	Translates an array of 256 glyph advance widths from their order in the PDF file into host encoding order.

1.3.3.4.2.7.1 IG_PDF_font_get_bbox

Gets a Type 3 font's bounding box, which is the smallest rectangle that would enclose every character in the font if they were overlaid and painted.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_bbox(
        HIG_PDF_FONT hFont,
        LPAT_PDF_FIXEDRECT lpBBox
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose bounding box is obtained.
lpBBox	LPAT_PDF_FIXEDRECT	The font's bounding box.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.7.2 IG_PDF_font_get_charset

Gets the font's character set.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_charset(
        HIG_PDF_FONT hFont,
        LPAT_INT lpCharSet
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose character set is obtained.
lpCharSet	LPAT_INT	The font's character set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

For non-Roman character set viewers, call IG_PDF_font_get_encoding_name() instead.

1.3.3.4.2.7.3 IG_PDF_font_get_cid_systeminfo

Gets Registry and Ordering information for a CIDFont.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_cid_systeminfo(
        HIG_PDF_FONT hFont,
        LPHIG_PDF_ATOM lphCIDSystemInfo
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose Registry and Ordering information is obtained.
lphCIDSystemInfo	LPHIG_PDF_ATOM	CIDFont's Registry and Ordering information.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function takes either a Type 0 font or descendant font (CIDType0 or CIDType2) as an argument. This information is always present for any Type 0 font; the actual registry ordering information is a part of the descendant font.

1.3.3.4.2.7.4 IG_PDF_font_get_cid_system_supplement

Gets the SystemSupplement number of a CIDFont.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_cid_system_supplement(
        HIG_PDF_FONT hFont,
        LPLONG lpCIDSystemSupplement
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose SystemSupplement field is obtained.
lpCIDSystemSupplement	LPLONG	The SystemSupplement field from the CIDFont.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.7.5 IG_PDF_font_get_descendant

Gets a Type 0 font's descendant, which may be a CIDType0 or CIDType2 font.

Declaration:

```
AT_ERRCOUNT ACCUAPI (
        HIG_PDF_FONT hFont,
        LPHIG_PDF_FONT lphFont
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose descendant is obtained.
lphFont	LPHIG_PDF_FONT	The font's descendant font.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.7.6 IG_PDF_font_get_encoding_index

Gets a font's encoding index.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_encoding_index(
        HIG_PDF_FONT hFont,
        LPLONG lpIndex
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose encoding index is obtained.
lpIndex	LPLONG	A font encoding index.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

For non-Roman character set viewers, it is not appropriate to call this function; call IG_PDF_font_get_encoding_name() instead.

1.3.3.4.2.7.7 IG_PDF_font_get_encoding_name

Gets a string representing a font's encoding.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_encoding_name(
        HIG_PDF_FONT hFont,
        LPBYTE* EncodingName
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose encoding name is obtained.
EncodingName	LPBYTE*	String representing the font's encoding.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Use IG_PDF_font_get_encoding_index() to get encoding information for Roman viewers.

1.3.3.4.2.7.8 IG_PDF_font_get_font_matrix

Gets a font's matrix, which specifies the transformation from character space to text space.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_font_matrix(
        HIG_PDF_FONT hFont,
        LPAT_PDF_FIXEDMATRIX lpMatrix
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose matrix is obtained.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Font's matrix.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This is only valid for Type 3 fonts.

1.3.3.4.2.7.9 IG_PDF_font_get_metrics

Gets a font's metrics, which provide the information needed to create a substitute multiple master font when the original font is unavailable.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_metrics(
        HIG_PDF_FONT hFont,
        LPAT_PDF_FONT_METRICS lpFontMetrics
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose metrics are obtained.
lpFontMetrics	LPAT_PDF_FONT_METRICS	The font's metrics.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.7.10 IG_PDF_font_get_name

Gets the name of a font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_name(
        HIG_PDF_FONT hFont,
        LPSTR buffer,
        LONG bufSize,
        LPLONG lpCharacterNum
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose name is obtained.
buffer	LPSTR	Buffer into which the font's name is stored. The client may pass NULL to obtain the buffer size, then call the method with a buffer of the appropriate size.
bufSize	LONG	Length of buffer, in bytes.
lpCharacterNum	LPLONG	The number of characters in the font name.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The behavior depends on the font type; for a Type 3 font it gets the value of the Name key in a PDF Font resource. For other types, it gets the value of the BaseFont key in a PDF font resource.

1.3.3.4.2.7.11 IG_PDF_font_get_subtype

Gets a font's subtype.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_subtype(
        HIG_PDF_FONT hFont,
        LPHIG_PDF_ATOM lphSubType
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose subtype is obtained.
lphSubType	LPHIG_PDF_ATOM	The font's subtype.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.7.12 IG_PDF_font_get_widths

Gets the advance width of every glyph in a font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_get_widths(
        HIG_PDF_FONT hFont,
        LPSHORT lpWidths
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose glyph advance widths are obtained.
lphSubType	LPSHORT	An array of glyph advance widths, measured in character space units. Unencoded code points will have a width of zero. For non-Roman character set viewers, an array for a single byte range (0 through 255).

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The advance width is the amount by which the current point advances when the glyph is drawn. The advance width may not correspond to the visible width of the glyph (for example, a glyph representing an accent mark might have an advance width of zero so that characters can be drawn under it). For this reason, the advance width cannot be used to determine the glyphs' bounding boxes. For non-Roman character set viewers, this function gets the width for a single byte range (0 through 255).

1.3.3.4.2.7.13 IG_PDF_font_is_embedded

Tests whether the specified font is embedded in the PDF file (that is, the font is stored as a font file, which is a stream embedded in the PDF file).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_is_embedded(
        HIG_PDF_FONT hFont,
        LPAT_PDF_BOOL lpbIsEmbedded
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font to test.
lpbIsEmbedded	LPAT_PDF_BOOL	Returns TRUE if the font is embedded in the file; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Only Type 1 and TrueType fonts can be embedded.

1.3.3.4.2.7.14 IG_PDF_font_set_metrics

Sets a font's metrics, which provide the information needed to create a substitute multiple master font when the original font is unavailable.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_set_metrics(
        HIG_PDF_FONT hFont,
        LPAT_PDF_FONT_METRICS lpFontMetrics
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose metrics are being set.
lpFontMetrics	LPAT_PDF_FONT_METRICS	Pointer to a AT_PDF_FONT_METRICS structure containing the font's metrics.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function can only be used on Type 1, multiple master Type 1, and TrueType fonts; it cannot be used on Type 3 fonts.

1.3.3.4.2.7.15 IG_PDF_font_translate_string

Translates a string from the hFont's encoding into host encoding.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_translate_string(
        HIG_PDF_FONT hFont,
        LPSTR inP,
        LPSTR outP,
        LONG len,
        LPAT_PDF_BOOL lpbTableExists
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font (and hence, encoding) that inP uses.
inP	LPSTR	The string to translate.
outP	LPSTR	The translated string. outP may point to the same buffer as inP to allow in-place translation.
len	LONG	The length of inP and outP.
lpbTableExists	LPAT_PDF_BOOL	Returns TRUE if an XlateTable exists in the font; FALSE otherwise. If no XlateTable exists in the font, outP is not written.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If any characters cannot be represented in host encoding, they are replaced with space characters. If no XlateTable exists in the font, the function returns FALSE and outP is not written. For non-Roman character set viewers, it is not appropriate to call this function.

1.3.3.4.2.7.16 IG_PDF_font_translate_to_host

Translates a string from the hFont's encoding to host encoding.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_translate_to_host(
        HIG_PDF_FONT hFont,
        LPSTR inP,
        LONG inLen,
        LPSTR outP,
        LONG outLen,
        LPLONG lpLen
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font used in the input string inP.
inP	LPSTR	The pointer to the string to translate.
inLen	LONG	The length of the inP buffer, in bytes.
outP	LPSTR	The pointer to the translated string.
outLen	LONG	The length of the outP buffer, in bytes.
lpLen	LPLONG	The number of bytes in the translated string outP.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.7.17 IG_PDF_font_translate_to_ucs

Translates a string from whatever encoding the hFont uses to Unicode encoding.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_translate_to_ucs(
        HIG_PDF_FONT hFont,
        LPSTR inP,
        LONG inLen,
        LPSTR outP,
        LONG outLen,
        LPLONG lpLen
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font of the input string inP.
inP	LPSTR	The pointer to the string to translate.
inLen	LONG	The length of the inP buffer, in bytes.
outP	LPSTR	The pointer to the translated string.
outLen	LONG	The length of the outP buffer, in bytes.
lpLen	LPLONG	The number of bytes in the translated string outP.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.7.18 IG_PDF_font_translate_widths

Translates an array of 256 glyph advance widths from their order in the PDF file into host encoding order.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_font_translate_widths(
        HIG_PDF_FONT hFont,
        LPSHORT inP,
        LPSHORT outP
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDF_FONT	The font whose glyph widths are translated.
inP	LPSHORT	Array of glyph advance widths to rearrange.
outP	LPSHORT	Rearranged array of glyph advance widths.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If the widths are already in host encoding order, the widths are merely copied. All un-encoded code points are given a width of zero. For non-Roman character set viewers, it is not appropriate to call this function.

1.3.3.4.2.8 HIG_PDF_LAYER

Handle to the PDF layer object which represents a named object whose state can be toggled in a User Interface to affect changes in visibility of content.

Members:

IG_PDF_layer_create	Creates new layer (optional-content group) object in the document.
IG_PDF_layer_get_name	Gets the layer name.
IG_PDF_layer_set_name	Sets the new layer name.
IG_PDF_layer_get_current_state	Gets the current ON-OFF state of the layer object.
IG_PDF_layer_set_current_state	Sets the current ON-OFF state of the layer object.
IG_PDF_layer_get_initial_state	Gets the initial ON-OFF state of the layer object.
IG_PDF_layer_set_initial_state	Sets the initial ON-OFF state of the layer object.
IG_PDF_layer_has_usage_info	Tests whether a layer object is associated with a Usage dictionary.
IG_PDF_layer_get_usage_info	Gets usage information from a layer object.
IG_PDF_layer_set_usage_info	Sets a Usage dictionary entry in a layer object.
IG_PDF_layer_get_intent	Gets the intent list for a layer.
IG_PDF_layer_set_intent	Sets the Intent entry in a layer's dictionary.
IG_PDF_layer_get_unique_id	Sets a unique ID.
IG_PDF_layer_release	Releases a layer object.
IG_PDF_layer_remove	Destroys layer (optional-content group) object.

1.3.3.4.2.8.1 IG_PDF_layer_create

Creates new layer (optional-content group) object in the document.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_create(
        HIG_PDF_DOC hDoc,
        LPSTR lpTextBuf,
        AT_DWORD nTextLen,
        LPHIG_PDF_LAYER lphLayer
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the group is used.
lpTextBuf	LPSTR	The name of the layer.
nTextLen	AT_DWORD	Length of the lpTextBuf.
lphLayer	LPHIG_PDF_LAYER	A pointer to memory that is overwritten with the value of the newly created group object handle.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.2 IG_PDF_layer_get_current_state

This function gets the current ON-OFF state of the layer object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_current_state(
        HIG_PDF_LAYER hLayer,
        LPAT_PDF_BOOL lpbState
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer whose current state is obtained.
lpbState	LPAT_PDF_BOOL	Returns TRUE if the state is ON; FALSE if it is OFF.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.3 IG_PDF_layer_get_initial_state

This function gets the initial ON-OFF state of the layer object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_initial_state(
        HIG_PDF_LAYER hLayer,
        LPAT_PDF_BOOL lpbState
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer whose initial state is obtained.
lpbState	LPAT_PDF_BOOL	Returns TRUE if the state is ON; FALSE if it is OFF.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.4 IG_PDF_layer_get_intent

This function gets the intent list for a layer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_intent(
        HIG_PDF_LAYER hLayer,
        LPHIG_PDF_ATOM lphIntent,
        UINT nSize,
        LPUINT lpnLen
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer object whose intent is obtained.
lphIntent	LPHIG_PDF_ATOM	Returns array of intent entries.
nSize	UINT	Maximal size of lphIntent.
lpnLen	LPUINT	Actual number of intent entries copied to lphIntent.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Intent is an atom value broadly describing the intended use, either View or Design. A layer's content is considered to be optional (that is, the layer's state is considered in its visibility) if any intent in its list matches an intent of the context. The intent list of the context is usually set from the intent list of the document configuration.

1.3.3.4.2.8.5 IG_PDF_layer_get_name

This function gets the layer name.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_name(
        HIG_PDF_LAYER hLayer,
        LPSTR lpTextBuf,
        UINT nBufSize,
        LPUINT lpnTextLen
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer whose name is obtained.
lpTextBuf	LPSTR	Pointer to a buffer for text string.
nBufSize	UINT	Maximum size of the buffer.
lpnTextLen	LPUINT	Actual length of the name string copied into the buffer.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.6 IG_PDF_layer_get_unique_id

Returns some 32-bit integer that is unique for all Layer objects.

Declaration:

```
IG_PDF_layer_get_unique_id(
        HIG_PDF_LAYER hLayer,
        LPUINT lpnUniqueId
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer object.
lpnUniqueId	LPUINT	The unique identifier.

Return Value:

A unique identifier of this Layer Object.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

It is guaranteed that there cannot be two Layer objects with the same UniqueIds. This can be used for Layer objects' identification.

1.3.3.4.2.8.7 IG_PDF_layer_get_usage_info

This function gets usage information from a layer object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_get_usage_info(
        HIG_PDF_LAYER hLayer,
        HIG_PDF_ATOM hUsageKey,
        LPHIG_PDF_BASOBJ lphUsageInfo
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer object whose usage information is obtained.
hUsageKey	HIG_PDF_ATOM	The usage key in the usage dictionary entry. Possible key values are: ● CreatorInfo ● Language ● Export ● Zoom ● Print ● View ● User ● PageElement
lphUsageInfo	LPHIG_PDF_BASOBJ	The usage information associated with the given key in the Usage dictionary for the layer, or a NULL if the operation fails (because the layer is malformed or has no dictionary, or because the dictionary has no entry corresponding to the given key).

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

A Usage dictionary entry provides more specific intended usage information than an intent entry.

1.3.3.4.2.8.8 IG_PDF_layer_has_usage_info

This function verifies whether a layer object is associated with a Usage dictionary.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_has_usage_info(
        HIG_PDF_LAYER hLayer,
        LPAT_PDF_BOOL lpbHasUsage
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer object.
lpbHasUsage	LPAT_PDF_BOOL	Returns TRUE if the layer has a Usage dictionary; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.9 IG_PDF_layer_release

Releases the native object and frees its memory.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_release(
        HIG_PDF_LAYER hLayer
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer object to release.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.10 IG_PDF_layer_remove

Destroys layer (optional-content group) object, but does not delete any content.

Declaration:

```
IG_PDF_layer_remove(
        HIG_PDF_LAYER hLayer
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	The layer object.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.11 IG_PDF_layer_set_current_state

This function sets the current ON-OFF state of the layer object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_current_state(
        HIG_PDF_LAYER hLayer,
        AT_PDF_BOOL bState
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer whose current state is set.
bState	AT_PDF_BOOL	New state.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.12 IG_PDF_layer_set_initial_state

This function sets the initial ON-OFF state of the layer object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_initial_state(
        HIG_PDF_LAYER hLayer,
        AT_PDF_BOOL bState
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer whose initial state is set.
bState	AT_PDF_BOOL	The new state.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.13 IG_PDF_layer_set_intent

This function sets the Intent entry in a layer's dictionary.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_intent(
        HIG_PDF_LAYER hLayer,
        LPHIG_PDF_ATOM lphIntent,
        UINT nLen
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer object whose intent is set.
lphIntent	LPHIG_PDF_ATOM	New Intent entry value, an array of intent entries (atoms).
nLen	UINT	Number of intent entries in lphIntent.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Intent is an atom value broadly describing the intended use, either View or Design. A layer's content is considered to be optional (that is, the layer's state is considered in its visibility) if any intent in its list matches an intent of the context. The intent list of the context is usually set from the intent list of the document configuration.

1.3.3.4.2.8.14 IG_PDF_layer_set_name

This function sets the new layer name.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_name(
        HIG_PDF_LAYER hLayer,
        LPSTR lpTextBuf,
        UINT nTextLen
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer whose name is set.
lpTextBuf	LPSTR	Pointer to a string buffer.
nBufSize	UINT	Length of the string buffer.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.8.15 IG_PDF_layer_set_usage_info

This function sets a Usage dictionary entry in a layer object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_layer_set_usage_info(
        HIG_PDF_LAYER hLayer,
        HIG_PDF_ATOM hUsageKey,
        HIG_PDF_BASOBJ hUsageInfo
);
```

Arguments:

Name	Type	Description
hLayer	HIG_PDF_LAYER	Layer object whose usage information is set.
hUsageKey	HIG_PDF_ATOM	The usage key in the usage dictionary entry. Possible key values are: ● CreatorInfo ● Language ● Export ● Zoom ● Print ● View ● User ● PageElement
hUsageInfo	HIG_PDF_BASOBJ	The usage information to associate with the key.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The entry associates usage information with an entry key for retrieval. If a dictionary does not exist, the method creates one.

A Usage dictionary entry provides more specific intended usage information than an intent entry.

The usage value can act as a kind of metadata, describing the sort of things that belong to the layer: for example, text in French, fine detail on a map, or a watermark. The usage values can also be used by the AutoState mechanism to make decisions about what layers should be on and what layers should be off.

1.3.3.4.2.9 HIG_PDF_PAGE

Handle to the PDF page object. A single page in the PDF representation of a document. A page contains a series of objects representing the objects drawn on the page (Graphic), a list of resources used in drawing the page, annotations (Annotation), an optional thumbnail image of the page, and the beads used in any articles that occur on the page.

Members:

IG_PDF_page_get_content	Creates HIG_PDE_CONTENT from HIG_PDF_PAGE.
IG_PDF_page_get_crop_box	Gets the crop box for the page.
IG_PDF_page_get_rotation	Gets the rotation for the page.
IG_PDF_page_make_color_separations	Separates hPage's DeviceN colorants into individual layers.
IG_PDF_page_set_content	Sets the page's PDF content back into the HIG_PDF_PAGE object, using the same compression filters with which the content was previously encoded.
IG_PDF_page_release_content	Decrements HIG_PDF_PAGE's PDF content internal reference count.
IG_PDF_page_get_annotation_count	Gets the number of annotations on a page.

1.3.3.4.2.9.1 IG_PDF_page_get_annotation_count

Gets the number of annotations on a page.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_get_annotation_count(
        HIG_PDF_PAGE hPage,
        LPUINT lpnCount
);
```

Arguments:

Name	Type	Description
hPage	HIG_PDF_PAGE	The page for which the number of annotations is obtained.
lpnCount	LPUINT	The number of annotations return value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Annotations associated with pop-up windows (such as strikeouts) are counted as two annotations. Widget annotations (form fields) are included in the count.

1.3.3.4.2.9.2 IG_PDF_page_get_content

Creates HIG_PDE_CONTENT from HIG_PDF_PAGE.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_get_content(
        HIG_PDF_PAGE hPage,
        LPHIG_PDE_CONTENT lpnContent
);
```

Arguments:

Name	Type	Description
hPage	HIG_PDF_PAGE	The page whose content object is acquired.
lpnContent	LPHIG_PDE_CONTENT	PDE content return value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The PDF content is cached, so that subsequent calls on the same page return the same PDF content.

1.3.3.4.2.9.3 IG_PDF_page_get_crop_box

Gets the crop box for the page.

Declaration:

```
AT_ERRCOUNT ACCUAPI i_IG_PDF_page_get_crop_box(
        HIG_PDF_PAGE hPage,
        LPAT_PDF_FIXEDRECT lpFixedBox
);
```

Arguments:

Name	Type	Description
hPage	HIG_PDF_PAGE	The page whose crop box is obtained.
lpFixedBox	LPAT_PDF_FIXEDRECT	The crop box.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.9.4 IG_PDF_page_get_rotation

Gets the rotation for the page.

Declaration:

```
AT_ERRCOUNT ACCUAPI i_IG_PDF_page_get_rotation(
        HIG_PDF_PAGE hPage,
        LPSHORT lpRotation
);
```

Arguments:

Name	Type	Description
hPage	HIG_PDF_PAGE	The page whose rotation is obtained.
lpRotation	LPSHORT	The rotation.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.9.5 IG_PDF_page_make_color_separations

Separates hPage's DeviceN colorants into individual layers; the resulting page is inserted in hDoc after the page with index nAfterPage.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_make_color_separations(
        HIG_PDF_PAGE hPage,
        LPHIG_PDF_ATOM pColorChannels,
        UINT nColorChannelsNum,
        HIG_PDF_DOC hDoc,
        LONG nAfterPage
);
```

Arguments:

Name	Type	Description
hPage	HIG_PDF_PAGE	The page whose content object is acquired.
pColorChannels	LPHIG_PDF_ATOM	Atom array containing the colorant names to separate. Example: "C", "M", "Y", "K", "PANTONE 300 C", etc.
nColorChannelsNum	UINT	Number of elements in pColorChannels.
hDoc	HIG_PDF_DOC	Output PDF document.
nAfterPage	LONG	The page number in the output PDF document after which the result page is inserted.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
HIG_PDF_ATOM* pColorChannels = new HIG_PDF_ATOM[2];
IG_PDF_atom_from_string("Y", &pColorChannels[0]);
IG_PDF_atom_from_string("PANTONE 300 C", &pColorChannels[1]);
nErrCount += IG_PDF_page_make_color_separations(GetCurPDFPage(), pColorChannels, 2,
GetPDFDoc(), nPageCount-2);
delete pColorChannels;
```

1.3.3.4.2.9.6 IG_PDF_page_release_content

Decrements HIG_PDF_PAGE's PDF content internal reference count.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_release_content(
        HIG_PDF_PAGE hPage
);
```

Arguments:

Name	Type	Description
hPage	HIG_PDF_PAGE	The page whose content object's use count is decremented.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The content is not automatically deleted when the reference count becomes zero - it remains in the cache until the cache slot is needed for another HIG_PDF_PAGE. Thus, you do not need to keep a content acquired for performance reasons.

1.3.3.4.2.9.7 IG_PDF_page_set_content

Sets the page's PDF content back into the HIG_PDF_PAGE object, using the same compression filters with which the content was previously encoded.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_page_set_content(
        HIG_PDF_PAGE hPage
);
```

Arguments:

Name	Type	Description
hPage	HIG_PDF_PAGE	The page whose content object is set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.10 HIG_PDF_STREAM

Handle to the PDF stream object. A data stream that may be a buffer in memory, or an arbitrary user-written procedure. Typically used to extract or provide data.

Members:

IG_PDF_stream_open_mem_for_read	Creates a read-only PDF stream from a memory-resident buffer.
IG_PDF_stream_read_CB_register	Creates a read-only PDF stream from an arbitrary data-producing procedure.
IG_PDF_stream_write_CB_register	Creates a PDF stream from an arbitrary data-producing procedure.
IG_PDF_stream_read	Creates a read-only PDF stream from an arbitrary data-producing procedure.
IG_PDF_stream_flush	Flushes any buffered data to the specified stream.
IG_PDF_stream_close	Closes the specified stream.

1.3.3.4.2.10.1 IG_PDF_stream_close

Closes the specified stream.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_close(
        HIG_PDF_STREAM hStream
);
```

Arguments:

Name	Type	Description
hStream	HIG_PDF_STREAM	The stream to close.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.10.2 IG_PDF_stream_flush

Flushes any buffered data to the specified stream.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_flush(
        HIG_PDF_STREAM hStream,
        LPLONG lpnResult
);
```

Arguments:

Name	Type	Description
hStream	HIG_PDF_STREAM	The stream to flush.
lpnResult	LPLONG	0 if successful; otherwise non-zero.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.10.3 IG_PDF_stream_open_mem_for_read

Creates a read-only PDF stream from a memory-resident buffer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_open_mem_for_read(
        LPSTR lpData,
        UINT nLen,
        LPHIG_PDF_STREAM lphStream
);
```

Arguments:

Name	Type	Description
lpData	LPSTR	Buffer containing the data to read into the stream. This data buffer must not be disposed of until the stream is closed.
nLen	UINT	Length of data, in bytes.
lphStream	LPHIG_PDF_STREAM	Handle to the new stream.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The stream is seek-able.

1.3.3.4.2.10.4 IG_PDF_stream_read

Creates a read-only PDF stream from an arbitrary data-producing procedure.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_read(
        HIG_PDF_STREAM hStream,
        LPSTR lpBuffer,
        LONG nItems,
        SHORT nItemSize,
        LPLONG lpnItemsRead
);
```

Arguments:

Name	Type	Description
hStream	HIG_PDF_STREAM	The stream from which data is read.
lpBuffer	LPSTR	Buffer into which data is written.
nItems	LONG	Number of items to read. The amount of data read into the memory buffer will be nItems ? nItemSize, unless an EOF is encountered first. The relative values of nItems and nItemSize really do not matter; the only thing that matters is their product. It is often convenient to set nItemSize to 1, so that nItems is the number of bytes to read.
nItemSize	SHORT	Number of bytes in an item in the stream.
lpnItemsRead	LPLONG	The number of items (not bytes) read.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The stream is not seek-able. lpfnReadProc is called when the client of the stream attempts to read data from it.

1.3.3.4.2.10.5 IG_PDF_stream_read_CB_register

Creates a read-only PDF stream from an arbitrary data-producing procedure.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_read_CB_register(
        LPFNIG_PDF_STREAM_PROC lpfnReadProc,
        LPVOID lpClientData,
        LPHIG_PDF_STREAM lphStream
);
```

Arguments:

Name	Type	Description
lpfnReadProc	LPFNIG_PDF_STREAM_PROC	User-supplied callback that supplies the stream's data.
lpClientData	LPVOID	Pointer to user-supplied data to pass to lpfnReadProc each time it is called.
lphStream	LPHIG_PDF_STREAM	Handle to the new stream.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The stream is not seek-able. lpfnReadProc is called when the client of the stream attempts to read data from it.

1.3.3.4.2.10.6 IG_PDF_stream_write_CB_register

Creates a PDF stream from an arbitrary data-producing procedure.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_stream_write_CB_register(
        LPFNIG_PDF_STREAM_PROC lpfnWriteProc,
        LPFNIG_PDF_STREAM_DESTROYPROC lpfnDestroyProc,
        LPVOID lpClientData,
        LPHIG_PDF_STREAM lphStream
);
```

Arguments:

Name	Type	Description
lpfnWriteProc	LPFNIG_PDF_STREAM_PROC	User-supplied callback that provides the data for the stream.
lpfnDestroyProc	LPFNIG_PDF_STREAM_DESTROYPROC	User-supplied callback that destroys the specified stream.
lpClientData	LPVOID	Pointer to user-supplied data to pass to lpfnWriteProc each time it is called.
lphStream	LPHIG_PDF_STREAM	Handle to the new stream.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The stream is not seek-able.

1.3.3.4.2.11 HIG_PDF_STYLE

Handle to the PDF style object; provides access to information about the fonts, font sizes, and colors used in a Word.

Members:

IG_PDF_style_get_color	Gets a style's color.
IG_PDF_style_get_font	Gets the specified style's font.
IG_PDF_style_get_font_size	Get a style's font size.
IG_PDF_style_delete	Deletes a PDF style object.

1.3.3.4.2.11.1 IG_PDF_style_delete

Deletes a PDF style object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_style_delete(
        HIG_PDF_STYLE hStyle
);
```

Arguments:

Name	Type	Description
hStyle	HIG_PDF_STYLE	Style object to delete.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.11.2 IG_PDF_style_get_color

Gets a style's color.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_style_get_color(
        HIG_PDF_STYLE hStyle,
        LPAT_PDF_COLORVALUE lpColor
);
```

Arguments:

Name	Type	Description
hStyle	HIG_PDF_STYLE	The style whose color is obtained.
lpColor	LPAT_PDF_COLORVALUE	The style's color.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.11.3 IG_PDF_style_get_font

Gets the specified style's font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_style_get_font(
        HIG_PDF_STYLE hStyle,
        LPHIG_PDF_FONT lphFont
);
```

Arguments:

Name	Type	Description
hStyle	HIG_PDF_STYLE	The style whose font is obtained.
lphFont	LPHIG_PDF_FONT	The font for the specified style.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.11.4 IG_PDF_style_get_font_size

Get a style's font size.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_style_get_font_size(
        HIG_PDF_STYLE hStyle,
        LPAT_PDF_FIXED lpnFontSize
);
```

Arguments:

Name	Type	Description
hStyle	HIG_PDF_STYLE	The style whose font size is obtained.
lpnFontSize	LPAT_PDF_FIXED	A style's font size.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.12 HIG_PDF_SYSENCODING

Handle to the system encoding object. Provides system encoding for a PDF file.

Members:

IG_PDF_sysencoding_create_from_base_name	Create an encoding object from base name.
IG_PDF_sysencoding_create_from_cmap_name	Create an encoding object from a PDF CMap name.
IG_PDF_sysencoding_create_from_code_page	Create an encoding object from a PDF CMap name.
IG_PDF_sysencoding_get_writing_mode	Returns writing mode in lpnWritingMode.
IG_PDF_sysencoding_is_identity	Returns in lpbResult TRUE for Identity-H or Identity-V encoding; FALSE otherwise.
IG_PDF_sysencoding_is_multibyte	Returns in lpbResult TRUE for CMap encoding; FALSE otherwise.
IG_PDF_sysencoding_release	Release an encoding object.

1.3.3.4.2.12.1 IG_PDF_sysencoding_create_from_base_name

Create an encoding object from base name.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_create_from_base_name(
        HIG_PDF_ATOM hBaseEncName,
        LPCSTR* lpDiffEnc,
        LPHIG_PDF_SYSENCODING lphSysEncoding
);
```

Arguments:

Name	Type	Description
hBaseEncName	HIG_PDF_ATOM	The base encoding. See Section 5.5.5 in the PDF Reference.
lpDiffEnc	LPCSTR*	Array of 256 const char* describing the differences from the encoding specified by hBaseEncName. May be NULL.
lphSysEncoding	LPHIG_PDF_SYSENCODING	New encoding object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.12.2 IG_PDF_sysencoding_create_from_cmap_name

Create an encoding object from a PDF CMap name.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_create_from_cmap_name(
        HIG_PDF_ATOM hCMapName,
        LPHIG_PDF_SYSENCODING lphSysEncoding
);
```

Arguments:

Name	Type	Description
hCMapName	HIG_PDF_ATOM	The CMap name.
lphSysEncoding	LPHIG_PDF_SYSENCODING	New encoding object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.12.3 IG_PDF_sysencoding_create_from_code_page

Create an encoding object from a code page.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_create_from_code_page(
        LONG nCodePage,
        SHORT nWritingMode,
        LPHIG_PDF_SYSENCODING lphSysEncoding
);
```

Arguments:

Name	Type	Description
nCodePage	LONG	The code page character-mapping construct. One of enumIGPDFCodePages values.
nWritingMode	SHORT	0 for horizontal writing, 1 for vertical writing.
lphSysEncoding	LPHIG_PDF_SYSENCODING	New encoding object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.12.4 IG_PDF_sysencoding_get_writing_mode

Returns writing mode in lpnWritingMode; 0 for horizontal writing, and 1 for vertical writing.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_get_writing_mode(
        HIG_PDF_SYSENCODING hSysEncoding,
        LPSHORT lpnWritingMode
);
```

Arguments:

Name	Type	Description
hSysEncoding	HIG_PDF_SYSENCODING	An encoding object.
lpnWritingMode	LPSHORT	0 for horizontal writing, and 1 for vertical writing.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.12.5 IG_PDF_sysencoding_is_identity

Returns in lpbResult TRUE for Identity-H or Identity-V encoding; FALSE otherwise.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_is_identity(
        HIG_PDF_SYSENCODING hSysEncoding,
        LPAT_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hSysEncoding	HIG_PDF_SYSENCODING	An encoding object.
lpbResult	LPAT_BOOL	TRUE for Identity-H or Identity-V encoding; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.12.6 IG_PDF_sysencoding_is_multibyte

Returns in lpbResult TRUE for CMap encoding; FALSE otherwise.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_is_multibyte(
        HIG_PDF_SYSENCODING hSysEncoding,
        LPAT_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hSysEncoding	HIG_PDF_SYSENCODING	An encoding object.
lpbResult	LPAT_BOOL	TRUE for CMap encoding; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.12.7 IG_PDF_sysencoding_release

Releases an encoding object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysencoding_release(
        HIG_PDF_SYSENCODING hSysEncoding
);
```

Arguments:

Name	Type	Description
hSysEncoding	HIG_PDF_SYSENCODING	An encoding object to release.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.13 HIG_PDF_SYSFONT

Handle to the system font object. A reference to a font installed in the host system. SysFont methods allow you to list the fonts available in the host system and to find a font in the system that matches a PDE Font, if it is present.

Members:

IG_PDF_sysfont_enumerate	Enumerates all of the system fonts with a user-supplied procedure.
IG_PDF_sysfont_find	Finds a system font that matches the requested attributes.
IG_PDF_sysfont_find_for_pdefont	Finds a system font that matches the requested hFont.
IG_PDF_sysfont_get_platform_data	Gets platform-specific data for use by user interface code.
IG_PDF_sysfont_get_attrs	Gets the attributes of a system font.
IG_PDF_sysfont_get_cid_system_info	Derives the registry, ordering, and supplement information of a multi-byte system font.
IG_PDF_sysfont_get_create_flags	This function obtains lpnFlags that can be passed to IG_PDE_font_create_from_sysfont_and_encoding.
IG_PDF_sysfont_get_info	Gets high-level information about a system font.
IG_PDF_sysfont_get_name	Gets the PostScript or TrueType styled name for a system font.
IG_PDF_sysfont_get_widths	Gets the widths of a single byte encoded system font.
IG_PDF_sysfont_release_platform_data	Releases platform-specific data for the specified hSysFont.
IG_PDF_sysfont_release	Releases a system font object.

1.3.3.4.2.13.1 IG_PDF_sysfont_enumerate

Enumerates all of the system fonts with a user-supplied procedure.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_enumerate(
        LPFNIG_PDF_SYSFONT_ENUMPROC lpfnEnumProc,
        LPVOID lpClientData
);
```

Arguments:

Name	Type	Description
lpfnEnumProc	LPFNIG_PDF_SYSFONT_ENUMPROC	User-supplied callback to call once for each system font. Enumeration continues until all fonts have been enumerated, or until lpfnEnumProc returns FALSE.
lpClientData	LPVOID	Pointer to user-supplied data to pass to lpfnEnumProc each time it is called.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The SysFont parameters must be copied during the enumeration if they are needed beyond the lpfnEnumProc.

Developers should not assume that the lpfnEnumProc will get called. If no system fonts are found, lpfnEnumProc is never called.

1.3.3.4.2.13.2 IG_PDF_sysfont_find

Finds a system font that matches the requested attributes.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_find(
        LPAT_PDE_FONTATTRS lpAttrs,
        LONG nFlags,
        LPHIG_PDF_SYSFONT lphSysFont
);
```

Arguments:

Name	Type	Description
lpAttrs	LPAT_PDE_FONTATTRS	Pointer to AT_PDE_FONTATTRS structure with the attributes of the font for which you are searching.
nFlags	LONG	Bit field comprised of enumIGPDFSysFontMatchFlags values. Passing zero matches font by name only.
lphSysFont	LPHIG_PDF_SYSFONT	The desired system font.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.13.3 IG_PDF_sysfont_find_for_pdefont

Finds a system font that matches the requested hFont.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_find_for_pdefont(
        HIG_PDE_FONT hFont,
        LONG nFlags,
        LPHIG_PDF_SYSFONT lphSysFont );
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	A PDE Font whose matching system font is found.
nFlags	LONG	Bit field comprised of enumIGPDFSysFontMatchFlags values. Passing zero matches font by name only.
lphSysFont	LPHIG_PDF_SYSFONT	The desired system font.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.13.4 IG_PDF_sysfont_get_attrs

Gets the attributes of a system font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_attrs(
        HIG_PDF_SYSFONT hSysFont,
        LPAT_PDE_FONTATTRS lpAttrs
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	A SysFont object referencing a system font.
lpAttrs	LPAT_PDE_FONTATTRS	Pointer to AT_PDE_FONTATTRS with the attributes of a system font.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The attributes will be returned in the buffer pointed to by lpAttrs.

This call can be expensive to execute, as it may involve parsing the font in order to determine attributes.

1.3.3.4.2.13.5 IG_PDF_sysfont_get_cid_system_info

Derives the registry, ordering, and supplement information of a multi-byte system font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_cid_system_info(
        HIG_PDF_SYSFONT hSysFont,
        LPHIG_PDF_ATOM lphRegistry,
        LPHIG_PDF_ATOM lphOrdering,
        LPLONG lpnSupplement
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	A SysFont object referencing a multi-byte system font.
lphRegistry	LPHIG_PDF_ATOM	The PDF atom representing the CID Font's registry information, as in "Adobe".
lphOrdering	LPHIG_PDF_ATOM	The PDF atom representing the CID Font's ordering information, for example, "Japan1".
lpnSupplement	LPLONG	The SystemSupplement field from the CID Font.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This information can be used to create a PDE Font from a system font.

1.3.3.4.2.13.6 IG_PDF_sysfont_get_create_flags

This function obtains lpnFlags that can be passed to IG_PDE_font_create_from_sysfont_and_encoding.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_create_flags(
        HIG_PDF_SYSFONT hSysFont,
        HIG_PDF_SYSENCODING hSysEncoding,
        LPLONG lpnFlags
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	A SysFont object.
hSysEncoding	HIG_PDF_SYSENCODING	A SysEncoding object.
lpnFlags	LPLONG	Create flags that can be passed to IG_PDE_font_create_from_sysfont_and_encoding. If the combination of hSysFont and hSysEncoding is not allowed, it is set to -1.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If the combination of hSysFont and hSysEncoding is not allowed, it is set to -1.

1.3.3.4.2.13.7 IG_PDF_sysfont_get_info

Gets high-level information about a system font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_info(
        HIG_PDF_SYSFONT hSysFont,
        LPAT_PDE_FONT_INFO lpInfo
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	A SysFont object referencing a system font whose information is obtained.
lpInfo		Pointer to AT_PDE_FONT_INFO structure to fill with font information for hSysFont.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.13.8 IG_PDF_sysfont_get_name

Gets the PostScript or TrueType styled name for a system font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_name(
        HIG_PDF_SYSFONT hSysFont,
        LPHIG_PDF_ATOM lphName
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	A SysFont object referencing a system font whose name is obtained.
lphName	LPHIG_PDF_ATOM	The PDF atom for the system font's name.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.13.9 IG_PDF_sysfont_get_platform_data

Gets platform-specific data for use by user interface code.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_platform_data(
        HIG_PDF_SYSFONT hSysFont,
        LPAT_PDF_SYSFONT_PLATDATA* lpPlatData
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	A SysFont object referencing a system font.
lpPlatData	LPAT_PDF_SYSFONT_PLATDATA*	Pointer to an AT_PDF_SYSFONT_PLATDATA containing information relating to a system font. Returns NULL if out of memory.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Must be released when finished by IG_PDF_sysfont_release_platform_data.

1.3.3.4.2.13.10 IG_PDF_sysfont_get_widths

Gets the widths of a single byte encoded system font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_get_widths(
        HIG_PDF_SYSFONT hSysFont,
        LPSHORT lpWidths,
        LPAT_PDF_FIXED mmDesignVector
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	A SysFont object referencing a system font whose widths are obtained.
lpWidths	LPSHORT	Pointer to widths array. lpWidths must have room for 256 entries.
mmDesignVector	LPAT_PDF_FIXED	If hSysFont is a multiple master font, points to the design vector, whose length must equal the number of design axes of hSysFont.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.13.11 IG_PDF_sysfont_release

Releases a system font object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_release(
        HIG_PDF_SYSFONT hSysFont
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	A system font to release.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.13.12 IG_PDF_sysfont_release_platform_data

Releases platform-specific data for the specified hSysFont.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_sysfont_release_platform_data(
        HIG_PDF_SYSFONT hSysFont,
        LPAT_PDF_SYSFONT_PLATDATA lpPlatData
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	A SysFont object referencing a system font.
lpPlatData	LPAT_PDF_SYSFONT_PLATDATA	Pointer to AT_PDF_SYSFONT_PLATDATA containing platform-specific data to release.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.14 HIG_PDF_WORD

Handle to the PDF word object. A word in a PDF file. Each word contains a sequence of characters in one or more styles (see Style).

Members:

IG_PDF_word_get_char_offset	Returns a word's character offset from the beginning of its page.
IG_PDF_word_get_char_style	Returns a PDF Style object for the specified style in a word.
IG_PDF_word_get_charquad	Gets the quad, expressed in user space coordinates, for a specific character from a word.
IG_PDF_word_get_length	Gets the number of bytes in a word.
IG_PDF_word_get_quad	Gets the specified word's quad, specified in user space coordinates.
IG_PDF_word_get_quad_count	Gets the number of quads in a word.
IG_PDF_word_get_string	Gets a word's text and also converts ligatures to their constituent characters.
IG_PDF_word_get_style_transition	Gets the locations of style transitions in a word.
IG_PDF_word_delete	Deletes a word object.

1.3.3.4.2.14.1 IG_PDF_word_delete

Deletes a word object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_delete(
        HIG_PDF_WORD hWord
);
```

Arguments:

Name	Type	Description
hWord	HIG_PDF_WORD	Word object to delete.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.14.2 IG_PDF_word_get_char_offset

Returns a word's character offset from the beginning of its page.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_char_offset(
        HIG_PDF_WORD hWord,
        LPWORD lpnCharOffset
);
```

Arguments:

Name	Type	Description
hWord	HIG_PDF_WORD	The word whose character offset is obtained.
lpnCharOffset	LPWORD	The word's character offset. On multi-byte systems, it points to the first byte.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.14.3 IG_PDF_word_get_char_style

Returns a PDF Style object for the specified style in a word.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_char_style(
        HIG_PDF_WORDFINDER hWordFinder,
        HIG_PDF_WORD hWord,
        LONG nIndex,
        LPHIG_PDF_STYLE lphStyle
);
```

Arguments:

Name	Type	Description
hWordFinder	HIG_PDF_WORDFINDER	A word finder object.
hWord	HIG_PDF_WORD	The word whose character style is obtained.
nIndex	LONG	The index of the style to obtain. The first style in a word has an index of zero.
lphStyle	LPHIG_PDF_STYLE	The obtained style in the word. Returns NULL if nIndex is greater than the number of styles in the word.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.14.4 IG_PDF_word_get_charquad

Gets the bounding quadrilateral, expressed in user space coordinates, for a specific character from a word.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_charquad(
        HIG_PDF_WORD hWord,
        AT_WORD nByteIndex,
        LPAT_PDF_FIXEDQUAD lpQuad,
        LPAT_PDF_BOOL lpbHasQuad
);
```

Arguments:

Name	Type	Description
hWord	HIG_PDF_WORD	The word to inspect.
nByteIndex	AT_WORD	The byte index of the character quad to obtain. The first character in a word has an index of zero. Use IG_PDF_word_get_length to identify the number of bytes in the word.
lpQuad	LPAT_PDF_FIXEDQUAD	Pointer to the character's quad, expressed in user-space coordinates. Upon successful completion, the memory referenced with this parameter is written with the character's quad.
lpbHasQuad	LPAT_PDF_BOOL	Pointer to memory that indicates whether the byte index has a quad. Upon successful completion, the memory referenced with this parameter is set to TRUE if the byte index has a quad; otherwise FALSE is set.

Return Value:

The number of ImageGear errors that occurred during this function call. If there are no errors, the return value is IGE_SUCCESS.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

See IG_PDF_word_get_quad_count for a description of a quad.

The quad's height is the height of the font's bounding box, not the height of the tallest character used in the word. The font's bounding box is determined by the glyphs in the font that extend farthest above and below the baseline; it often extends somewhat above the top of "A" and below the bottom of "y."

The quad's width is determined from the characters actually present in the word.

As an example, the quads for the words "AWAY" and "away" have the same height, but generally do not have the same width unless the font is a mono-spaced font (a font in which all characters have the same width).

1.3.3.4.2.14.5 IG_PDF_word_get_length

Gets the number of bytes in a word.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_length(
        HIG_PDF_WORD hWord,
        LPWORD lpnLength
);
```

Arguments:

Name	Type	Description
hWord	HIG_PDF_WORD	The word whose character count is obtained.
lpnLength	LPWORD	The number of characters in the word.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function also works on non-Roman systems.

1.3.3.4.2.14.6 IG_PDF_word_get_quad

Gets the specified word's quad, specified in user space coordinates.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_quad(
        HIG_PDF_WORD hWord,
        SHORT nIndex,
        LPAT_PDF_FIXEDQUAD lpQuad,
        LPAT_PDF_BOOL lpbHasQuad
);
```

Arguments:

Name	Type	Description
hWord	HIG_PDF_WORD	The word whose quad is obtained.
nIndex	SHORT	The index of the quad to obtain. The first quad in a word has an index of zero.
lpQuad	LPAT_PDF_FIXEDQUAD	Pointer to the word's quad, specified in user-space coordinates.
lpbHasQuad	LPAT_PDF_BOOL	TRUE if the word has the specified quad; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

See IG_PDF_word_get_quad_count for a description of a quad.

The quad's height is the height of the font's bounding box, not the height of the tallest character used in the word. The font's bounding box is determined by the glyphs in the font that extend farthest above and below the baseline; it often extends somewhat above the top of "A" and below the bottom of "y."

The quad's width is determined from the characters actually present in the word.

As an example, the quads for the words "AWAY" and "away" have the same height, but generally do not have the same width unless the font is a mono-spaced font (a font in which all characters have the same width).

1.3.3.4.2.14.7 IG_PDF_word_get_quad_count

Gets the number of quads in a word.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_quad_count (
        HIG_PDF_WORD hWord,
        LPWORD lpnQuadCount
);
```

Arguments:

Name	Type	Description
hWord	HIG_PDF_WORD	The word whose quad count is obtained.
lpnQuadCount	LPWORD	The number of quads in the word.

Remarks:

A quad is a quadrilateral bounding a contiguous piece of a word. Every word has at least one quad. A word has more than one quad, for example, if it is hyphenated and split across multiple lines or if the word is set on a curve rather than on a straight line.

Supported Raster Image Formats:

This function does not process image pixels.

Return Value:

Error count.

1.3.3.4.2.14.8 IG_PDF_word_get_string

Gets a word's text and also converts ligatures to their constituent characters.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_string(
        HIG_PDF_WORD hWord,
        LPCHAR lpString,
        LONG nLen
);
```

Arguments:

Name	Type	Description
hWord	HIG_PDF_WORD	The word whose string is obtained.
lpString	LPCHAR	The word string. The encoding of the string is the encoding used by the PDF WordFinder that supplied the PDF Word. For instance, if IG_PDF_doc_create_wordfinder_ucs is used to create the word finder, this function returns only Unicode.
nLen	LONG	Length of string, in bytes. Up to nLen characters of word will be copied into lpString. If lpString is long enough, it will be null-terminated.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The string to return includes any word break characters (such as space characters) that follow the word, but not any that precede the word. The characters that are treated as word breaks are defined in the outEncInfo parameter of IG_PDF_doc_create_wordfinder function.

This function produces a string in whatever encoding the PDF Word uses, for both Roman and non-Roman systems.

1.3.3.4.2.14.9 IG_PDF_word_get_style_transition

Gets the locations of style transitions in a word.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_word_get_style_transition(
        HIG_PDF_WORD hWord,
        LPSHORT lpTransTbl,
        SHORT nSize,
        LPSHORT lpnStTrCount
);
```

Arguments:

Name	Type	Description
hWord	HIG_PDF_WORD	The word whose style transition list is obtained.
lpTransTbl	LPSHORT	(Filled by the method) Array of style transitions. Each element is the character offset in word where the style changes. The offset specifies the first character in the word that has the new style. The first character in a word has an offset of zero.
nSize	SHORT	Number of entries that lpTransTbl can hold. The word is searched only until this number of style transitions has been found.
lpnStTrCount	LPSHORT	Number of style transition offsets copied to lpTransTbl.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Every word has at least one style transition, at character position zero in the word.

1.3.3.4.2.15 HIG_PDF_WORDFINDER

Handle to the PDF word finder object. Extracts words from a PDF file, and enumerates the words on a single page or on all pages in a document.

Members:

IG_PDF_wordfinder_acquire_wordlist	Finds all words on the specified page.
IG_PDF_wordfinder_release_wordlist	Releases the word list for a given page.
IG_PDF_wordfinder_get_word	Gets the word in the word list obtained using IG_PDF_wordfinder_acquire_wordlist.
IG_PDF_wordfinder_delete	Deletes a word finder.

1.3.3.4.2.15.1 IG_PDF_wordfinder_acquire_wordlist

Finds all words on the specified page.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_wordfinder_acquire_wordlist(
        HIG_PDF_WORDFINDER hWordFinder,
        LONG nPageNumber,
        LPLONG lpnWordCount
);
```

Arguments:

Name	Type	Description
hWordFinder	HIG_PDF_WORDFINDER	The word finder used to acquire the word list.
nPageNumber	LONG	The page number for which words are found. First page is 0.
lpnWordCount	LPLONG	The number of words found on the page.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Only words within or partially within the page's crop box are enumerated. Words outside the crop box are skipped.

There can be only one word list in existence at a time; clients must release the previous word list, using IG_PDF_wordfinder_release_wordlist, before creating a new one.

1.3.3.4.2.15.2 IG_PDF_wordfinder_delete

Deletes a word finder.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_wordfinder_delete(
        HIG_PDF_WORDFINDER hWordFinder
);
```

Arguments:

Name	Type	Description
hWordFinder	HIG_PDF_WORDFINDER	Word finder object to delete.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Use this function when you are done extracting text in a file.

1.3.3.4.2.15.3 IG_PDF_wordfinder_get_word

Gets the word in the word list obtained using IG_PDF_wordfinder_acquire_wordlist.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_wordfinder_get_word (
        HIG_PDF_WORDFINDER hWordFinder,
        WORD nFlags,
        LONG nIndex,
        LPHIG_PDF_WORD lphWord
);
```

Arguments:

Name	Type	Description
hWordFinder	HIG_PDF_WORDFINDER	The word finder object.
nFlags	WORD	Word-finding options. Must be an OR of one or more of enumIGPDFWordFlags.
nIndex	LONG	The index of the word to obtain. The first word on a page has an index of zero. Words are counted in PDF order.
lphWord	HIG_PDF_WORD	A handle to the word object. Returns NULL when the end of the list is reached.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.2.15.4 IG_PDF_wordfinder_release_wordlist

Releases the word list for a given page.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDF_wordfinder_release_wordlist(
        HIG_PDF_WORDFINDER hWordFinder,
        LONG nPageNumber
);
```

Arguments:

Name	Type	Description
hWordFinder	HIG_PDF_WORDFINDER	The word finder object.
nPageNumber	LONG	The page number for which the word list is released.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Use this to release a list created by IG_PDF_wordfinder_acquire_wordlist when you are done using this list.

1.3.3.4.3 Page Editing Objects and Elements

Objects described in this Section provide easy access to PDF page contents. With this group of objects, you can treat a page's contents as a list of objects rather than having to manipulate the content stream's PDF marking operators. PDE objects are meant to be used in conjunction with the General and Base Object methods for manipulating PDF documents.

The PDE objects are split in two groups: PDE elements and PDE objects. A page display list is represented as a PDE object that contains PDE elements. The page's content is usually treated as a list of PDE elements. Each PDE element is a path, text, image, form, or a marked content place or container of PDE Elements. Using the PDE Objects and functions you can add or remove objects inside a PDE Object. You can also change attributes of Elements in a PDE Object, such as a bounding box, a text font, or a clipping path.

The PDE objects are used to create the PDE elements and provide specific information for the elements. For example, PDE Image, which is an element, usually has or is created based on a PDE ColorSpace, which is an object. PDE text, which is an element, usually has or is created based on a PDE Font, which is an object.

This API is meant to be used in conjunction with basic and general objects.

The following table contains the objects and their descriptions supported by the ImageGear PDF component:

Page Editing Objects and Elements	
HIG_PDE_OBJECT	Object - base interface for all the PDE objects.
HIG_PDE_CONTENT	Content - modifiable content of a PDF page, which contains elements. Content may be obtained from an existing page or from a Form XObject.
HIG_PDE_COLORSPACE	ColorSpace - a reference to a color space used on a page in a PDF file. The color space is part of the graphics state attributes of a PDE Element.
HIG_PDE_FONT	Font - A reference to a font used on a page in a PDF file. It may be equated with a font in the system.
HIG_PDE_ELEMENT	Element - base interface for the elements of a page display list (PDE content) and for clip objects. The general PDE element methods allow you to get and set general element properties.
HIG_PDE_CLIP	Clip - a list of elements containing a list of Paths and Texts that describe a clip state. Clips can be created and built up with PDE Clip methods. Any PDE Element object can have Clip associated with it. Clip objects can contain PDE Containers and PDE Groups to an arbitrary level of nesting. This allows PDE Containers to be used to mark clip objects. PDE Groups inside PDE Clips that contain at least one PDE Text and no PDE Paths have a special meaning. All PDE Text objects contained in such a PDE Group are considered to be part of the same BT/ET block. This means that the union of these PDE Texts makes up a single clipping path-as opposed to the intersection of the PDE Texts. See Section 5.3 in the PDF reference for more information about BT/ET block.
HIG_PDE_CONTAINER	Container - a group of elements on a page in a PDF file. In the PDF file, containers are delimited by Marked Content BMC/EMC or BDC/EMC pairs. Every container has a Marked Content tag associated with it. In addition to grouping a set of elements, a BDC/EMC pair specifies a property list to be associated with the grouping. Thus a container corresponding to a BDC/EMC pair also has a property list dictionary associated with it. See Section 10.5 in the PDF reference for more information about Marked Content operators.
HIG_PDE_FORM	Form - an element that corresponds to an instance of XObject Form on a page (or other containing stream such as another XObject Form or annotation form). The context associated with this instance includes the actual stream that represents the XObject Form and the initial conditions of the graphics state. The latter consists of the transformation matrix, initial color values, and so forth. It is possible to have two Forms that refer to the same XObject Form. The forms will exist at different places on the same page, depending on the transformation matrix. They may also have different colors or line stroking parameters. In the case of a transparency group, the opacity is specified in the gstate. Within a Form, each element has its own gstate (or is a container, place, or group object). These gstates are independent of the parent Form gstate. Form elements may have their own opacity. Content may be obtained from a Form to edit the form's display list.
HIG_PDE_GROUP	Group - an in-memory representation of objects in Content. It has no state and is not represented in any way in a content stream (that is, Content). When used in a Clip, this object is used to associate Text objects into a single clipping object.
HIG_PDE_IMAGE	Image - an element that contains an Image XObject or in-line image. You can associate data or a stream with an image.
HIG_PDE_PATH	Path - an element that contains a path. Path objects can be stroked, filled, and/or serve as a clipping path.

HIG_PDE_PLACE	Place - an element that marks a place on a page in a PDF file. In a PDF file, a place is represented by the MP or DP Marked Content operators. Marked content is useful for adding structure information to a PDF file. For instance, a drawing program may want to mark a point with information, such as the start of a path of a certain type. Marked content provides a way to retain this information in the PDF file. A DP operator functions the same as the MP operator and, in addition, allows a property list dictionary to be associated with a place.
HIG_PDE_POSTSCRIPT	PostScript - an element representing in-line or XObject pass-through PostScript object. XObject PostScripts are listed in page XObject resources.
HIG_PDE_SOFTMASK	SoftMask - an object for creating and manipulating a soft mask in a PDF file.
HIG_PDE_SHADING	Shading - an element that represents smooth shading.
HIG_PDE_TEXT	Text - an element representing text. It is a container for text as show strings or as individual characters. Each sub-element may have different graphics state properties. However, the same clip applies to all sub-elements of a Text. Also, the char path of a Text can be used to represent a clip.
HIG_PDE_TEXTITEM	TextItem - a PDE element representing a text object.
HIG_PDE_XGROUP	XGroup - a transparency (XGroup) resource.
HIG_PDE_XOBJECT	XObject - an element representing an arbitrary XObject.

1.3.3.4.3.1 HIG_PDE_OBJECT

Handle to abstract PDE object. Base interface for all the PDE objects.

Members:

IG_PDE_object_get_type	Gets the type of an object.

1.3.3.4.3.1.1 IG_PDE_object_get_type

Gets the type of an object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_object_get_type(
        HIG_PDE_OBJECT hObject,
        LPLONG lpnType
);
```

Arguments:

Name	Type	Description
hObject	HIG_PDE_OBJECT	The object whose type is obtained.
lpnType	LPLONG	Type return value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.2 HIG_PDE_CLIP

Handle to the PDE clip object. A clip is a list of elements containing a list of Paths and Texts that describe a clip state. Clips can be created and built up with PDE Clip methods. Any PDE Element object can have Clip associated with it. Clip objects can contain PDE Containers and PDE Groups to an arbitrary level of nesting. This allows PDE Containers to be used to mark clip objects.

PDE Groups inside PDE Clips that contain at least one PDE Text and no PDE Paths have a special meaning. All PDE Text objects contained in such a PDE Group are considered to be part of the same BT/ET block. This means that the union of these PDE Texts makes up a single clipping path, as opposed to the intersection of the PDE Texts.

See Section 5.3 in the PDF reference for more information about BT/ET block.

Members:

IG_PDE_clip_create	Creates an empty clip object.
IG_PDE_clip_clone	Makes a deep copy of a PDE Clip object.
IG_PDE_clip_add_element	Adds an element to a clip path.
IG_PDE_clip_get_element	Gets an element from a clip object.
IG_PDE_clip_remove_elements	Removes one or more elements from a clip object.
IG_PDE_clip_get_element_count	Gets the number of top-level elements in a clip object.
IG_PDE_clip_enumerate_elements	For a given PDE Clip, enumerates all of the PDE Elements in a flattened manner.

1.3.3.4.3.2.1 IG_PDE_clip_create

Creates an empty clip object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_create(
        LPHIG_PDE_CLIP lphClip
);
```

Arguments:

Name	Type	Description
lphClip	LPHIG_PDE_CLIP	The newly created clip object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This represents a clipping object that has no affect on elements that refer to it.

Call IG_PDE_element_release to dispose of the object.

1.3.3.4.3.2.2 IG_PDE_clip_clone

Makes a deep copy of a PDE Clip object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_clone(
        HIG_PDE_CLIP hClip,
        LPHIG_PDE_CLIP lphCloneClip
);
```

Arguments:

Name	Type	Description
hClip	HIG_PDE_CLIP	The clipping path to copy.
lphCloneClip	LPHIG_PDE_CLIP	The deep copy of hClip.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.2.3 IG_PDE_clip_add_element

Adds an element to a clip path.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_add_element(
        HIG_PDE_CLIP hClip,
        LONG nAfterIndex,
        HIG_PDE_ELEMENT hElement
);
```

Arguments:

Name	Type	Description
hClip	HIG_PDE_CLIP	The clip path to which an element is added.
nAfterIndex	LONG	The index after which to add hElement. Use IG_PDE_BEFORE_FIRST to insert an element at the beginning of the clip object.
hElement	HIG_PDE_ELEMENT	The element added, which may be a PDE Path, a PDE Text, a PDE Container, a PDE Group or a PDE Place object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.2.4 IG_PDE_clip_get_element

Gets an element from a clip object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_get_element(
        HIG_PDE_CLIP hClip,
        LONG nIndex,
        LPHIG_PDE_ELEMENT lphElement
);
```

Arguments:

Name	Type	Description
hClip	HIG_PDE_CLIP	The clip object from which an element is obtained.
nIndex	LONG	Index of element to get from clip.
lphElement	LPHIG_PDE_ELEMENT	The element from the clip object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.2.5 IG_PDE_clip_remove_elements

Removes one or more elements from a clip object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_remove_elements(
        HIG_PDE_CLIP hClip,
        LONG nIndex,
        LONG nCount
);
```

Arguments:

Name	Type	Description
hClip	HIG_PDE_CLIP	The clip object from which an element is removed.
nIndex	LONG	First element to remove.
nCount	LONG	Number of elements to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.2.6 IG_PDE_clip_get_element_count

Gets the number of top-level elements in a clip object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_get_element_count(
        HIG_PDE_CLIP hClip,
        LPUINT lpnCount
);
```

Arguments:

Name	Type	Description
hClip	HIG_PDE_CLIP	The clip object to examine.
lpnCount	LPUINT	Number of path and charpath elements in clip. If clip contains PDE Groups, this function returns the top-level PDE Path, PDE Text, PDE Container, PDE Group, or PDE Place object. Use IG_PDE_clip_enumerate_elements to see only the PDE Path and PDE Text objects. PDEGroup is not a persistent object. You cannot save to PDF and re-get group objects.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Top-level elements may be a path or char-path, a marked content container or place, or a group.

Paths are represented as PDE Path objects; char-paths are represented as PDE Text objects.

1.3.3.4.3.2.7 IG_PDE_clip_enumerate_elements

For a given PDE Clip, enumerates all of the PDE Elements in a flattened manner.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_clip_enumerate_elements (
        HIG_PDE_CLIP hClip,
        LPFNIG_PDE_CLIP_ENUMPROC lpfnEnumProc,
        LPVOID lpClientData,
        LPAT_PDF_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hClip	HIG_PDE_CLIP	The PDEClip to enumerate.
lpfnEnumProc	LPFNIG_PDE_CLIP_ENUMPROC	Called with each flattened element. Enumeration continues until all elements have been enumerated, or until lpfnEnumProc returns FALSE.
lpClientData	LPVOID	Pointer to user-supplied data to pass to lpfnEnumProc each time it is called.
lpbResult	LPAT_PDF_BOOL	Returns value of lpfnEnumProc. TRUE if successful; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

PDE Containers and PDE Groups nested in the PDE Clip will not be handed back, but any PDE Paths and PDE Texts nested in them will be. Additionally, PDE Place objects inside the PDE Clip are not returned.

1.3.3.4.3.3 HIG_PDE_COLORSPACE

Handle to the PDE color space object. A reference to a color space used on a page in a PDF file. The color space is part of the graphics state attributes of a PDE Element.

Members:

IG_PDE_colorspace_create	Creates a new color space object of the specified type.
IG_PDE_colorspace_get_base_name	Gets the name of the base color space.
IG_PDE_colorspace_get_base_color_components	Gets the number of components in the base color space of an indexed color space.
IG_PDE_colorspace_get_ctable	Gets the component information for an indexed color space.
IG_PDE_colorspace_get_hival	Gets the highest index for the color look-up table for an indexed color space.
IG_PDE_colorspace_get_name	Gets the name of a color space object.
IG_PDE_colorspace_get_color_components	Calculates the number of components in a color space.
IG_PDE_colorspace_release	Releases a color space object.

1.3.3.4.3.3.1 IG_PDE_colorspace_create

Creates a new color space object of the specified type.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_create(
        HIG_PDF_ATOM hFamily,
        LPAT_PDE_COLORDATA lpColorData,
        LPHIG_PDE_COLORSPACE lphColorSpace
);
```

Arguments:

Name	Type	Description
hFamily	HIG_PDF_ATOM	Supports the following PDF color spaces: • Device-dependent names: DeviceCMYK, DeviceGray, DeviceN, or DeviceRGB. • Device-independent names: CalGray, CalRGB, Lab, or ICCBased. • Special names: Indexed, Pattern, or Separation.
lpColorData	LPAT_PDE_COLORDATA	Color data for the type of color space you want to create.
lphColorSpace	LPHIG_PDE_COLORSPACE	Handle to the new color space.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.3.2 IG_PDE_colorspace_get_base_name

Gets the name of the base color space.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_base_name (
        HIG_PDE_COLORSPACE hColorSpace,
        LPHIG_PDF_ATOM lphBaseName
);
```

Arguments:

Name	Type	Description
hColorSpace	HIG_PDE_COLORSPACE	The base color space.
lphBaseName	LPHIG_PDF_ATOM	The atom for the name of the base color space.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This is a helper routine for indexed color spaces.

1.3.3.4.3.3.3 IG_PDE_colorspace_get_base_color_components

Gets the number of components in the base color space of an indexed color space.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_base_color_components (
        HIG_PDE_COLORSPACE hColorSpace,
        LPLONG lpnColorComponents
);
```

Arguments:

Name	Type	Description
hColorSpace	HIG_PDE_COLORSPACE	The base color space.
lpnColorComponents	LPLONG	Number of components in hColorSpace.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

For example, for [/Indexed /DeviceRGB...], the number of components is 3.

1.3.3.4.3.3.4 IG_PDE_colorspace_get_ctable

Gets the component information for an indexed color space.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_ctable (
        HIG_PDE_COLORSPACE hColorSpace,
        LPBYTE lpColorTable
);
```

Arguments:

Name	Type	Description
hColorSpace	HIG_PDE_COLORSPACE	The color space whose component information table is obtained.
lpColorTable	LPBYTE	The color look-up table, which is nColorComponents * (nHiVal + 1) bytes long, where nColorComponents = number of components in the base hColorSpace. Each entry in the table contains nColorComponents bytes, and the table is indexed 0 to nHiVal, where nHiVal is the highest index in the color table. The table is indexed from 0 to nHiVal, thus the table contains nHiVal + 1 entries.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.3.5 IG_PDE_colorspace_get_hival

Gets the highest index for the color look-up table for an indexed color space.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_hival (
        HIG_PDE_COLORSPACE hColorSpace,
        LPLONG lpnHiVal
);
```

Arguments:

Name	Type	Description
hColorSpace	HIG_PDE_COLORSPACE	An indexed color space.
lpnHiVal	LPLONG	The highest index (nHiVal) in the color look-up table.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Since the color table is indexed from zero to nHiVal, the actual number of entries is nHiVal + 1.

1.3.3.4.3.3.6 IG_PDE_colorspace_get_name

Gets the name of a color space object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_name (
        HIG_PDE_COLORSPACE hColorSpace,
        LPHIG_PDF_ATOM lphName
);
```

Arguments:

Name	Type	Description
hColorSpace	HIG_PDE_COLORSPACE	A color space object.
lphName	LPHIG_PDF_ATOM	The color space object's name. Supports the following PDF color spaces: ● Device-dependent names: DeviceCMYK, DeviceGray, DeviceN, or DeviceRGB. ● Device-independent names: CalGray, CalRGB, Lab, or ICCBased. ● Special names: Indexed, Pattern, or Separation.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.3.7 IG_PDE_colorspace_get_color_components

Calculates the number of components in a color space.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_get_color_components (
        HIG_PDE_COLORSPACE hColorSpace,
        LPLONG lnpColorComponents
);
```

Arguments:

Name	Type	Description
hColorSpace	HIG_PDE_COLORSPACE	A color space object.
lpnColorComponents	LPLONG	Number of components in hColorSpace. • DeviceGray, CalGray, Separation: Returns 1. • DeviceRGB, CalRGB: Returns 3. • DeviceCMYK, Lab: Returns 4. • DeviceN, ICCBased: Returns the number of components dependent on the specific color space object. • Indexed: Returns 1. Use IG_PDE_colorspace_get_base_color_components to get the number of components in the base color space.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.3.8 IG_PDE_colorspace_release

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_colorspace_release(
        HIG_PDE_COLORSPACE hColorSpace
);
```

Arguments:

Name	Type	Description
hColorSpace	HIG_PDE_COLORSPACE	Color space object to release.

Description:

Releases a color space object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.4 HIG_PDE_CONTAINER

Handle to the PDE container object. A container is a group of elements on a page in a PDF file. In the PDF file, containers are delimited by Marked Content BMC/EMC or BDC/EMC pairs. Every container has a Marked Content tag associated with it. In addition to grouping a set of elements, a BDC/EMC pair specifies a property list to be associated with the grouping. Thus a container corresponding to a BDC/EMC pair also has a property list dictionary associated with it.

See Section 10.5 in the PDF reference for more information about Marked Content operators.

Members:

IG_PDE_container_create	Creates a container object.
IG_PDE_container_get_content	Gets the PDE Content for hContainer.
IG_PDE_container_set_content	Sets the content for a container.
IG_PDE_container_get_dictionary	Gets the Marked Content dictionary for a container.
IG_PDE_container_set_dictionary	Changes the Marked Content dictionary for a container.
IG_PDE_container_get_mctag	Gets the Marked Content tag for a container.
IG_PDE_container_set_mctag	Sets the Marked Content tag for a hContainer.

1.3.3.4.3.4.1 IG_PDE_container_create

Creates a container object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_create(
        HIG_PDF_ATOM hMCTag,
        HIG_PDF_BASOBJ hDictionary,
        AT_PDF_BOOL bIsInline,
        LPHIG_PDE_CONTAINER lphContainer
);
```

Arguments:

Name	Type	Description
hMCTag	HIG_PDF_ATOM	Tag name for the container.
hDictionary	HIG_PDF_BASOBJ	Optional Marked Content dictionary for the container.
bIsInline	AT_PDF_BOOL	If TRUE, emits container into the page content stream inline.
lphContainer	LPHIG_PDE_CONTAINER	The newly created container object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose of the object.

1.3.3.4.3.4.2 IG_PDE_container_get_content

Gets the PDE Content for hContainer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_get_content(
        HIG_PDE_CONTAINER hContainer,
        LPHIG_PDE_CONTENT lphContent
);
```

Arguments:

Name	Type	Description
hContainer	HIG_PDE_CONTAINER	A container whose content is obtained.
lphContent	LPHIG_PDE_CONTENT	The PDE Content for hContainer.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.4.3 IG_PDE_container_set_content

Sets the content for a container.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_set_content(
        HIG_PDE_CONTAINER hContainer,
        HIG_PDE_CONTENT hContent
);
```

Arguments:

Name	Type	Description
hContainer	HIG_PDE_CONTAINER	A container whose content is set.
hContent	HIG_PDE_CONTENT	The content of hContainer.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The existing PDE Content is released by this function.

1.3.3.4.3.4.4 IG_PDE_container_get_dictionary

Gets the Marked Content dictionary for a container.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_get_dictionary(
        HIG_PDE_CONTAINER hContainer,
        LPHIG_PDF_BASOBJ lphDictionary,
        LPAT_PDF_BOOL lpbIsInline,
        LPAT_PDF_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hContainer	HIG_PDE_CONTAINER	A container.
lphDictionary	LPHIG_PDF_BASOBJ	Marked Content dictionary for hContainer. NULL if hContainer has no Marked Content dictionary.
lpbIsInline	LPAT_PDF_BOOL	TRUE if the dictionary is inline; FALSE otherwise. Undefined if hContainer has no Marked Content dictionary.
lpbResult	LPAT_PDF_BOOL	TRUE if hContainer has a Marked Content dictionary; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.4.5 IG_PDE_container_set_dictionary

Changes the Marked Content dictionary for a container.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_set_dictionary(
        HIG_PDE_CONTAINER hContainer,
        HIG_PDF_BASOBJ hDictionary,
        AT_PDF_BOOL bIsInline
);
```

Arguments:

Name	Type	Description
hContainer	HIG_PDE_CONTAINER	A container whose dictionary is changed.
hDictionary	HIG_PDF_BASOBJ	Marked Content dictionary being set into hContainer.
bIsInline	AT_PDF_BOOL	If TRUE, the dictionary is emitted inline.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.4.6 IG_PDE_container_get_mctag

Gets the Marked Content tag for a container.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_get_mctag (
        HIG_PDE_CONTAINER hContainer,
        LPHIG_PDF_ATOM lphMCTag
);
```

Arguments:

Name	Type	Description
hContainer	HIG_PDE_CONTAINER	A container.
lphMCTag	LPHIG_PDF_ATOM	Marked Content tag of hContainer. Returns IG_PDF_ATOM_NULL if hContainer has no Marked Content tag.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.4.7 IG_PDE_container_set_mctag

Sets the Marked Content tag for a hContainer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_container_set_mctag (
        HIG_PDE_CONTAINER hContainer,
        HIG_PDF_ATOM hMCTag
);
```

Arguments:

Name	Type	Description
hContainer	HIG_PDE_CONTAINER	A container to tag.
hMCTag	HIG_PDF_ATOM	Marked Content tag.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.5 HIG_PDE_CONTENT

Handle to the PDE content object. Modifiable content of a PDF page, which contains elements. Content may be obtained from an existing page or from a Form XObject.

Members:

IG_PDE_content_create	Creates empty PDEContent.
IG_PDE_content_get_element	Obtains requested element from content.
IG_PDE_content_add_element	Inserts an element into content.
IG_PDE_content_remove_element	Removes an element from content.
IG_PDE_content_get_element_count	Gets the number of elements in a content.
IG_PDE_content_get_default_color_space	Gets a default color space from hContent.
IG_PDE_content_get_attrs	Gets the attributes of a content.

1.3.3.4.3.5.1 IG_PDE_content_create

This function creates empty PDEContent.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_create(
        LPHIG_PDE_CONTENT lphContent
);
```

Arguments:

Name	Type	Description
lphContent	LPHIG_PDE_CONTENT	The created PDEContent.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.5.2 IG_PDE_content_get_element

Obtains requested element from content.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_get_element(
        HIG_PDE_CONTENT hContent,
        LONG nIndex,
        LPHIG_PDE_ELEMENT lphElement
);
```

Arguments:

Name	Type	Description
hContent	HIG_PDE_CONTENT	Content to obtain.
nIndex	LONG	Index of element to obtain.
lphElement	LPHIG_PDE_ELEMENT	PDE element return value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.5.3 IG_PDE_content_add_element

Inserts an element into content.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_add_element(
        HIG_PDE_CONTENT hContent,
        LONG nAfterIndex,
        HIG_PDE_ELEMENT hElement
);
```

Arguments:

Name	Type	Description
hContent	HIG_PDE_CONTENT	Content to which hElement is added.
nAfterIndex	LONG	Location after which hElement is added. Should be IG_PDE_BEFORE_FIRST to add to the beginning of the display list.
hElement	HIG_PDE_ELEMENT	The element to add to the content.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.5.4 IG_PDE_content_remove_element

Removes an element from content.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_remove_element(
        HIG_PDE_CONTENT hContent,
        LONG nIndex
);
```

Arguments:

Name	Type	Description
hContent	HIG_PDE_CONTENT	Content to remove.
nIndex	LONG	Index of element to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.5.5 IG_PDE_content_get_element_count

Gets the number of elements in a content.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_get_element_count(
        HIG_PDE_CONTENT hContent,
        LPUINT lpnCount
);
```

Arguments:

Name	Type	Description
hContent	HIG_PDE_CONTENT	Content.
lpnCount	LPUINT	The number of elements return value .

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.5.6 IG_PDE_content_get_default_color_space

Gets a default color space from hContent.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_get_default_color_space(
        HIG_PDE_CONTENT hContent,
        HIG_PDF_ATOM hColorSpaceName,
        LPHIG_PDE_COLORSPACE lphDefaultColorSpace
);
```

Arguments:

Name	Type	Description
hContent	HIG_PDE_CONTENT	Content to whose default color space is obtained.
hColorSpaceName	HIG_PDF_ATOM	An atom for the name of the desired color space. Must be an atom for one of DefaultRGB, DefaultCMYK, or DefaultGray.
lphDefaultColorSpace	LPHIG_PDE_COLORSPACE	The desired color space in hContent. Returns NULL if hColorSpaceName does not correspond to a known default, such as DefaultRGB.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

See Section 4.5.4 in the PDF Reference for more information about default color spaces.

1.3.3.4.3.5.7 IG_PDE_content_get_attrs

Gets the attributes of a content.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_content_get_attrs (
        HIG_PDE_CONTENT hContent,
        LPAT_PDE_CONTENTATTRS lpAttrs
);
```

Arguments:

Name	Type	Description
hContent	HIG_PDE_CONTENT	Content whose attributes are obtained
lpAttrs	LPAT_PDE_CONTENTATTRS	Pointer to an AT_PDE_CONTENTATTRS structure to fill with the attributes of the content.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.6 HIG_PDE_ELEMENT

Handle to the PDE element object. Base interface for the elements of a page display list (PDE content) and for clip objects. The general PDE element methods allow you to get and set general element properties.

Members:

IG_PDE_element_get_type	Gets the type of an element.
IG_PDE_element_clone	Makes a copy of an element.
IG_PDE_element_is_at_point	Tests whether a point is on an element.
IG_PDE_element_is_at_rect	Tests whether any part of a rectangle is on an element.
IG_PDE_element_get_bbox	Gets the bounding box for an element.
IG_PDE_element_get_clip	Gets the current clip for an element.
IG_PDE_element_get_gstate	Gets the graphics state information for an element.
IG_PDE_element_set_gstate	Sets the graphics state information for an element.
IG_PDE_element_get_matrix	Gets the transformation matrix for an element.
IG_PDE_element_set_matrix	Sets the transformation matrix for an element.
IG_PDE_element_get_dictionary	Returns a Dictionary (OCMD object).
IG_PDE_element_get_unique_id	Returns some 32bit integer that is unique for all Element objects.
IG_PDE_element_has_gstate	Tests if hElement has a graphics state information.
IG_PDE_element_release	Release the specified element.

1.3.3.4.3.6.1 IG_PDE_element_get_type

Gets the type of an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_type(
        HIG_PDE_ELEMENT hElement,
        LPLONG lpnType
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	The element whose type is obtained.
lpnType	LPLONG	Type return value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.6.2 IG_PDE_element_clone

Makes a copy of an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_clone(
        HIG_PDE_ELEMENT hElement,
        LONG nFlags,
        LPHIG_PDE_ELEMENT lphCloneElement
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	The element to copy.
nFlags	LONG	Bit field of enumIGPDEElementCopyFlags.
lphCloneElement	LPHIG_PDE_ELEMENT	A copy of hElement.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The caller is responsible for releasing the copy with IG_PDE_element_release.

1.3.3.4.3.6.3 IG_PDE_element_is_at_point

Tests whether a point is on an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_is_at_point(
        HIG_PDE_ELEMENT hElement,
        LPAT_PDF_FIXEDPOINT lpPoint,
        LPAT_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	The element to test. ● If hElement is Text or an Image, it uses the bounding box of the element to make the check. ● If the hElement is a Path and it is stroked, it checks if the point is on the path. ● If the hElement is a Path and it is filled, it checks if the point is in the fill area, taking into consideration whether it is filled using the non-zero winding number rule or the even-odd rule.
lpPoint	LPAT_PDF_FIXEDPOINT	The point, specified in user space coordinates.
lpbResult	LPAT_BOOL	TRUE if the point is on the element; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.6.4 IG_PDE_element_is_at_rect

Tests whether any part of a rectangle is on an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_is_at_rect(
        HIG_PDE_ELEMENT hElement,
        LPAT_PDF_FIXEDRECT lpRect,
        LPAT_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	The element to test. ● If hElement is a PDE Text or PDE Image, it uses the bounding box of the PDE Element to make the check. ● If hElement is a PDE Path and it is stroked, it checks if the rectangle is on the path. ● If hElement is a PDE Path and it is filled, it checks if the rectangle is in the fill area, taking into consideration whether it is filled using the non-zero winding number rule or the even-odd rule.
lpRect	LPAT_PDF_FIXEDRECT	The rectangle, specified in user space coordinates.
lpbResult	LPAT_BOOL	True if any part of the rectangle is on the element; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.6.5 IG_PDE_element_get_bbox

Gets the bounding box for an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_bbox(
        HIG_PDE_ELEMENT hElement,
        LPAT_PDF_FIXEDRECT lpBBox
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	An element whose bounding box is obtained.
lpBBox	LPAT_PDF_FIXEDRECT	Pointer to an AT_PDF_FIXEDRECT structure specifying the bounding box of hElement, specified in user space coordinates.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The returned bounding box is guaranteed to encompass the element, but is not guaranteed to be the smallest box that could contain the element. For example, for an arc, lpBBox encloses the Bezier control points, not just the curve itself.

1.3.3.4.3.6.6 IG_PDE_element_get_clip

Gets the current clip for an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_clip(
        HIG_PDE_ELEMENT hElement,
        LPHIG_PDE_CLIP lphClip
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	An element whose clip is obtained. A clip may be shared by many elements. Use care when modifying a clip. Copy it first if you want to modify the clip for a specific element.
lphClip	LPHIG_PDE_CLIP	Clip object for hElement.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.6.7 IG_PDE_element_set_clip

Sets the clip for an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_set_clip(
        HIG_PDE_ELEMENT hElement,
        HIG_PDE_CLIP hClip
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	An element whose clip is set.
hClip	HIG_PDE_CLIP	The clip to set for hElement.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.6.8 IG_PDE_element_get_gstate

Gets the graphics state information for an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_gstate(
        HIG_PDE_ELEMENT hElement,
        LPAT_PDE_GRAPHICSTATE lpGstate
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	An element whose graphics state is obtained.
lpGstate	LPAT_PDE_GRAPHICSTATE	Pointer to AT_PDE_GRAPHICSTATE structure that contains graphics state information for hElement. This graphics state information may contain PDE objects for color spaces or an ExtGState. They are not acquired by this function. For a PDE Image, only the ExtGState value is used for images. For indexed images, the fill color space and values are categorized in the PDE Image object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function is only valid for PDE Form, PDE Image, PDE Path, and PDE Shading elements.

Non-NULL objects in the graphic state, such as the fill and stroke color spaces, have their reference counts incremented by this function. Be sure to release these non-NULL objects when disposing of lpGstate.

1.3.3.4.3.6.9 IG_PDE_element_set_gstate

Sets the graphics state information for an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_set_gstate(
        HIG_PDE_ELEMENT hElement,
        LPAT_PDE_GRAPHICSTATE lpGstate
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	An element whose graphics state is set.
lpGstate	LPAT_PDE_GRAPHICSTATE	Pointer to AT_PDE_GRAPHICSTATE structure with graphics state information to set for hElement.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function is valid only for PDE Form, PDE Image, PDE Path, and PDE Shading elements.

1.3.3.4.3.6.10 IG_PDE_element_get_matrix

Gets the transformation matrix for an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_get_matrix(
        HIG_PDE_ELEMENT hElement,
        LPAT_PDF_FIXEDMATRIX lpMatrix
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	An element whose transformation matrix is obtained.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX that holds a transformation matrix for hElement. If hElement is a text object, returns the identity matrix.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This matrix provides the transformation from user space to device space for the element. If there is no cm operator (concatmatrix) in the page stream, the matrix is the identity matrix.

1.3.3.4.3.6.11 IG_PDE_element_set_matrix

Sets the transformation matrix for an element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_set_matrix(
        HIG_PDE_ELEMENT hElement,
        LPAT_PDF_FIXEDMATRIX lpMatrix
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	An element whose transformation matrix is set.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX that holds the transformation matrix to set for hElement.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The element may not be a PDE Container, a PDE Group, a PDE Place, or a PDE Text.

1.3.3.4.3.6.12 IG_PDE_element_get_dictionary

Returns a Dictionary (OCMD object) that is associated with this Element; or, if no Dictionary is associated, returns NULL.

Declaration:

```
IG_PDE_element_get_dictionary(
        HIG_PDE_ELEMENT hElement,
        LPHIG_PDF_DICTIONARY lphOCMD
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	The Source Element.
lphOCMD	LPHIG_PDF_DICTIONARY	The returned Dictionary (OCMD object).

Return Value:

Associated Dictionary, or NULL if no Dictionary is associated.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.6.13 IG_PDE_element_get_unique_id

Returns some 32-bit integer that is unique for all Element objects.

Declaration:

```
IG_PDE_element_get_unique_id(
        HIG_PDE_ELEMENT hElement,
        LPLONG lpnId
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	Element object.
lpnId	LPLONG	The unique identifier.

Return Value:

An unique identifier of this Element Object.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

It is guaranteed that there cannot be two Element objects with the same UniqueIds.

Can be used for Element objects' identification.

1.3.3.4.3.6.14 IG_PDE_element_has_gstate

Tests if hElement has a graphics state information.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_has_gstate(
        HIG_PDE_ELEMENT hElement,
        LPAT_PDF_BOOL lpbHasGstate
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	An element whose graphics state is checked.
lpbHasGstate	LPAT_PDF_BOOL	Returns TRUE if the element has a graphics state; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.6.15 IG_PDE_element_release

Release the specified element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_element_release(
        HIG_PDE_ELEMENT hElement
);
```

Arguments:

Name	Type	Description
hElement	HIG_PDE_ELEMENT	The element to release.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7 HIG_PDE_FONT

Handle to the PDE font object. A reference to a font used on a page in a PDF file. It may be equated with a font in the system.

Members:

IG_PDE_font_create	Creates a new PDE Font from the specified parameters.
IG_PDE_font_create_from_sysfont	Creates a PDE Font corresponding to a font in the system.
IG_PDE_font_create_from_sysfont_and_encoding	Create a PDE Font from hSysFont and hSysEncoding.
IG_PDE_font_create_from_sysfont_with_params	Used to obtain a PDE Font corresponding to a font in the system.
IG_PDE_font_get_attrs	Gets the attributes for a font object.
IG_PDE_font_create_tounicode_now	Creates the /ToUnicode table.
IG_PDE_font_create_widths_now	Creates width entries for font.
IG_PDE_font_embed_now	Embeds font stream.
IG_PDE_font_embed_now_dont_subset	Embeds the given hFont inside hDoc without creating a subset.
IG_PDE_font_get_create_need_flags	Returns flags indicating what needs to be done to make hFont complete.
IG_PDE_font_get_codebyte_count	Gets the number of bytes comprising the next code in a string of single or multi-byte character codes.
IG_PDE_font_get_onebyte_encoding	Gets an array of delta encodings for the given one byte PDE Font.
IG_PDE_font_get_sysencoding	Gets the system encoding object associated with a font object.
IG_PDE_font_get_sysfont	Gets the system font object associated with a font object.
IG_PDE_font_get_widths	Gets the widths for a font object.
IG_PDE_font_get_widths_now	Gets a Type0 font's width information for only the characters used in the file.
IG_PDE_font_is_embedded	Tests whether a font is an embedded font in the document in which it was created.
IG_PDE_font_is_multibyte	Tests whether a font contains any multi-byte characters.
IG_PDE_font_set_sysencoding	Sets the system encoding object associated with a font object.
IG_PDE_font_set_sysfont	Sets the system font object to be used with a font object that does not currently have a system font associated with it.
IG_PDE_font_subset_now	Subsets a given PDE Font in hDoc.
IG_PDE_font_sum_widths	Gets the sum to the widths of nTextLen characters from a string of single or multi-byte characters.
IG_PDE_font_translate_glyphids_to_unicode	Translates a string to Unicode values.
IG_PDE_font_release	Releases PDE font object.

1.3.3.4.3.7.1 IG_PDE_font_create

Creates a new PDE Font from the specified parameters.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create(
        LPAT_PDE_FONTATTRS lpAttrs,
        LONG nFirstCharIndex,
        LONG nLastCharIndex,
        LPSHORT lpWidths,
        LPSTR* lpEncoding,
        HIG_PDF_ATOM hEncodingBaseName,
        HIG_PDF_STREAM hFontStm,
        LONG nLen1,
        LONG nLen2,
        LONG nLen3,
        LPHIG_PDE_FONT lphFont
);
```

Arguments:

Name	Type	Description
lpAttrs	LPAT_PDE_FONTATTRS	Pointer to AT_PDE_FONTATTRS for the font attributes.
nFirstCharIndex	LONG	First character index for the widths array, lpWidths.
nLastCharIndex	LONG	Last character index for the widths array, lpWidths.
lpWidths	LPSHORT	Widths array.
lpEncoding	LPSTR*	Array of 256 glyph names specifying the custom encoding. If any pointer is NULL, no encoding information is written for that entry.
hEncodingBaseName	HIG_PDF_ATOM	Encoding base name if the encoding is a custom encoding. If encoding is NULL, encodingBaseName is used as the value of the encoding, and must be one of WinAnsiEncoding, MacRomanEncoding, or MacExpertEncoding. If no encoding value is desired, use IG_PDF_ATOM_NULL.
hFontStm	HIG_PDF_STREAM	Stream with font information.
nLen1	LONG	Length in bytes of the ASCII portion of the Type 1 font file after it has been decoded. For other font formats, such as TrueType or CFF, only Len1 is used, and it is the size of the font.
nLen2	LONG	Length in bytes of the encrypted portion of the Type 1 font file after it has been decoded.
nLen3	LONG	Length in bytes of the portion of the Type 1 font file that contains the 512 zeros, plus the clear-to-mark operator, plus any following data.
lphFont	LPHIG_PDE_FONT	The specified PDE font.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The PDE Font may be represented as an embedded font (a FontFile entry in the font descriptor of the PDF file). To create a PDE Font that is stored as an embedded font, the FontFile stream may be passed in hFontStm, and the nLen1, nLen2, and nLen3 parameters contain the Length1, Length2, and Length3 values of the FontFile stream attributes dictionary. See Section 5.8 in the PDF Reference for more information about embedded fonts.

Call IG_PDE_font_release to dispose of the returned font object when finished with it.

1.3.3.4.3.7.2 IG_PDE_font_create_from_sysfont

Creates a PDE Font corresponding to a font in the system.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create_from_sysfont(
        HIG_PDF_SYSFONT hSysFont,
        LONG nCreateFlags,
        HIG_PDF_ATOM hSnapshotName,
        LPAT_PDF_FIXED mmDesignVec,
        LPHIG_PDE_FONT lphFont
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	PDF system font object referencing a system font.
nCreateFlags	LONG	Indicates whether to embed the font and whether to subset the font. Must be one of enumIGPDEFontCreateFlags. If you want to subset a font, set both the IG_PDE_FONT_CREATE_EMBEDDED and IG_PDE_FONT_WILL_SUBSET flags.
hSnapshotName	HIG_PDF_ATOM	Name to be associated with this particular instantiation of the PDE Font.
mmDesignVec	LPAT_PDF_FIXED	Multiple master font design vector.
lphFont	LPHIG_PDE_FONT	The PDE Font corresponding to hSysFont.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If the font is a multiple master font, mmDesignVec points to the design vector, whose length must equal the number of design axes of the font.

The enumIGPDEFontCreateFlags flags IG_PDE_FONT_CREATE_EMBEDDED and IG_PDE_FONT_WILL_SUBSET must both be set in order to subset a font.

If you create a PDE Font that is a subset, call IG_PDE_font_subset_now on this font afterwards.

Call IG_PDE_font_release to dispose of the returned font object when finished with it.

1.3.3.4.3.7.3 IG_PDE_font_create_from_sysfont_and_encoding

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create_from_sysfont_and_encoding(
        HIG_PDF_SYSFONT hSysFont,
        HIG_PDF_SYSENCODING hSysEncoding,
        HIG_PDF_ATOM hUseThisBaseFont,
        LONG nCreateFlags,
        LPHIG_PDE_FONT lphFont
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	PDF system font object referencing a system font.
hSysEncoding	HIG_PDF_SYSENCODING	A PDF SysEncoding object.
hUseThisBaseFont	HIG_PDF_ATOM	The base font. An error will be set if the base font name passed is a subset name (XXXXXX+FontName) or an empty string.
nCreateFlags	LONG	One of the enumIGPDEFontCreateFlags values.
lphFont	LPHIG_PDE_FONT	The new PDE Font object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Create a PDE Font from hSysFont and hSysEncoding. If it fails, it returns an error. Users can call IG_PDF_sysfont_get_create_flags to see if the combination of hSysFont and hSysEncoding makes sense.

Call IG_PDE_font_release to dispose of the returned font object when finished with it.

1.3.3.4.3.7.4 IG_PDE_font_create_from_sysfont_with_params

Used to obtain a PDE Font corresponding to a font in the system.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create_from_sysfont_with_params(
        HIG_PDF_SYSFONT hSysFont,
        LPAT_PDE_FONT_CREATEFROMSYSFONTPARAMS lpParams,
        LPHIG_PDE_FONT lphFont
);
```

Arguments:

Name	Type	Description
hSysFont	HIG_PDF_SYSFONT	PDF system font object referencing a system font.
lpParams	LPAT_PDE_FONT_CREATEFROMSYSFONTPARAMS	Pointer to the parameters structure.
lphFont	LPHIG_PDE_FONT	The new PDE Font object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_font_release to dispose of the returned font object when finished with it.

1.3.3.4.3.7.5 IG_PDE_font_get_attrs

Gets the attributes for a font object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_attrs(
        HIG_PDE_FONT hFont,
        LPAT_PDE_FONTATTRS lpAttrs
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font whose attributes are found.
lpAttrs	LPAT_PDE_FONTATTRS	Font attributes return value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.6 IG_PDE_font_create_tounicode_now

This function creates the /ToUnicode table.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create_tounicode_now(
        HIG_PDE_FONT hFont,
        HIG_PDF_DOC hDoc
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font for which to create /ToUnicode table.
hDoc	HIG_PDF_DOC	The container document.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The user can check the return value of IG_PDE_font_get_create_need_flags to see if calling of IG_PDE_font_create_tounicode_now is needed.

1.3.3.4.3.7.7 IG_PDE_font_create_widths_now

This function creates width entries for font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_create_widths_now(
        HIG_PDE_FONT hFont,
        HIG_PDF_DOC hDoc
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font for which to create width entries.
hDoc	HIG_PDF_DOC	The container document.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The user can check the return value of IG_PDE_font_get_create_need_flags to see if calling of IG_PDE_font_create_widths_now is needed.

1.3.3.4.3.7.8 IG_PDE_font_embed_now

This function embeds font stream.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_embed_now(
        HIG_PDE_FONT hFont,
        HIG_PDF_DOC hDoc );
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font to embed.
hDoc	HIG_PDF_DOC	The container document.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The user can check the return value of IG_PDE_font_get_create_need_flags to see if calling of IG_PDE_font_embed_now is needed.

1.3.3.4.3.7.9 IG_PDE_font_embed_now_dont_subset

Embeds the given hFont inside hDoc without creating a subset.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_embed_now_dont_subset(
        HIG_PDE_FONT hFont,
        HIG_PDF_DOC hDoc );
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font to embed.
hDoc	HIG_PDF_DOC	The container document.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Use this function instead of IG_PDE_font_embed_now if you created font with the IG_PDE_FONT_WILL_SUBSET flag but changed your mind.

1.3.3.4.3.7.10 IG_PDE_font_get_create_need_flags

This function returns flags indicating what needs to be done to make hFont complete.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_create_need_flags(
        HIG_PDE_FONT hFont,
        LPLONG lpnFlags
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font object.
lpnFlags	LPLONG	A value corresponding to enumIGPDEFontCreateNeedFlags.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

IG_PDE_FONT_CREATE_NEED_WIDTHS can be cleared by IG_PDE_font_create_widths_now.

IG_PDE_FONT_CREATE_NEED_TO_UNICODE can be cleared by IG_PDE_font_create_tounicode_now.

IG_PDE_FONT_CREATE_NEED_EMBED can be cleared by IG_PDE_font_embed_now.

1.3.3.4.3.7.11 IG_PDE_font_get_codebyte_count

Gets the number of bytes comprising the next code in a string of single or multi-byte character codes.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_codebyte_count(
        HIG_PDE_FONT hFont,
        LPBYTE lpText,
        LONG nTextLen,
        LPUINT lpnCount
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font object.
lpText	LPBYTE	Pointer into a string of characters.
nTextLen	LONG	The length, in bytes, of the string of characters, starting with the character pointed to by lpText.
lpnCount	LPUINT	Number of bytes in the next character code pointed to by lpText.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.12 IG_PDE_font_get_onebyte_encoding

Gets an array of delta encodings for the given one byte PDE Font.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_onebyte_encoding(
        HIG_PDE_FONT hFont,
        LPHIG_PDF_ATOM lphEncodingDelta,
        LPAT_BOOL lpbGotEncodingDelta
);
```

Arguments:

Name	**Type**	**Description**
hFont	HIG_PDE_FONT	PDE font object.
lphEncodingDelta	LPHIG_PDF_ATOM	Pointer to an atom array that is filled with the delta encodings for font. Each entry is the atom for a glyph name that differs from the base encoding. See Section 5.5.5 in the PDF Reference for more information about font encodings. The array must be allocated to hold 256 entries.
lpbGotEncodingDelta	LPAT_BOOL	TRUE if encodingDelta is filled; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.13 IG_PDE_font_get_sysencoding

Gets the system encoding object associated with a font object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_sysencoding(
        HIG_PDE_FONT hFont,
        LPHIG_PDF_SYSENCODING lphSysEncoding
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font whose system encoding is found.
lphSysEncoding	LPHIG_PDF_SYSENCODING	The system encoding object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.14 IG_PDE_font_get_sysfont

Gets the system font object associated with a font object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_sysfont(
        HIG_PDE_FONT hFont,
        LPHIG_PDF_SYSFONT lphSysFont
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font whose system font is found.
lphSysFont	LPHIG_PDF_SYSFONT	The system font object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.15 IG_PDE_font_get_widths

Gets the widths for a font object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_widths(
        HIG_PDE_FONT hFont,
        LPSHORT lpWidths
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font whose widths are found.
lpWidths	LPSHORT	Pointer to widths array. lpWidths must have room for 256 values. The widths are returned in character space (1000 EM units). An EM is a typographic unit of measurement equal to the size of a font. To convert to text space, divide the value returned by 1000. To convert to user space, multiply the text space value by the font size.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.16 IG_PDE_font_get_widths_now

Gets a Type0 font's width information for only the characters used in the file.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_get_widths_now(
        HIG_PDE_FONT hFont,
        HIG_PDF_DOC hDoc
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font whose widths are found.
hDoc	HIG_PDF_DOC	The container document.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call this routine when the font was created with the IG_PDE_FONT_DEFER_WIDTHS flag but without the IG_PDE_FONT_CREATE_EMBEDDED flag (if the font is to be embedded, call IG_PDE_font_subset_now, which also gets the width info).

1.3.3.4.3.7.17 IG_PDE_font_is_embedded

Tests whether a font is an embedded font in the document in which it was created.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_is_embedded(
        HIG_PDE_FONT hFont,
        LPAT_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font object.
lpbResult	LPAT_BOOL	TRUE if the font is embedded; FALSE if it is not, or if it was created in one document and embedded in a different document.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.18 IG_PDE_font_is_multibyte

Tests whether a font contains any multi-byte characters.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_is_multibyte(
        HIG_PDE_FONT hFont,
        LPAT_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font object.
lpbResult	LPAT_BOOL	TRUE if the font contains any multi-byte characters; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.19 IG_PDE_font_set_sysencoding

Sets the system encoding object associated with a font object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_set_sysencoding(
        HIG_PDE_FONT hFont,
        HIG_PDF_SYSENCODING hSysEncoding
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE Font whose system encoding is set.
hSysEncoding	HIG_PDF_SYSENCODING	The new system encoding object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Changing the system encoding may produce unexpected results.

1.3.3.4.3.7.20 IG_PDE_font_set_sysfont

Sets the system font object to be used with a font object that does not currently have a system font associated with it.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_set_sysfont(
        HIG_PDE_FONT hFont,
        HIG_PDF_SYSFONT hSysFont
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE Font whose system font is set.
hSysFont	HIG_PDF_SYSFONT	The new system font object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.21 IG_PDE_font_subset_now

Subsets a given PDE Font in hDoc.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_subset_now(
        HIG_PDE_FONT hFont,
        HIG_PDF_DOC hDoc
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font to subset.
hDoc	HIG_PDF_DOC	The document whose font is subset.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

If you created font with IG_PDE_font_create_from_sysfont, you must have set both the IG_PDE_FONT_CREATE_EMBEDDED and IG_PDE_FONT_WILL_SUBSET set in the flags parameter to be able to subset the font.

1.3.3.4.3.7.22 IG_PDE_font_sum_widths

Gets the sum to the widths of nTextLen characters from a string of single or multi-byte characters.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_sum_widths(
        HIG_PDE_FONT hFont,
        LPBYTE lpText,
        LONG nTextLen,
        LPLONG lpnSum
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font object.
lpText	LPBYTE	Pointer into a string of characters.
nTextLen	LONG	Number of characters in the string.
lpnSum	LPLONG	Width of text string in EM space. (In EM space, the width of "M" is about 1000 EM units).

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.7.23 IG_PDE_font_translate_glyphids_to_unicode

Translates a string to Unicode values.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_translate_glyphids_to_unicode(
        HIG_PDE_FONT hFont,
        LPBYTE lpText,
        LONG nTextLen,
        LPBYTE lpUniText,
        LONG nUniTextLen,
        LPLONG lpnResult
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font object.
lpText	LPBYTE	The string to convert.
nTextLen	LONG	The length of lpText, in bytes.
lpUniText	LPBYTE	Buffer to hold the translated string.
nUniTextLen	LONG	The size of the lpUniText buffer.
lpnResult	LPLONG	0 if the string was successfully translated. If lpUniText is too small for the translated string, it returns the number of bytes required.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The PDE Font must have a /ToUnicode table.

1.3.3.4.3.7.24 IG_PDE_font_release

Releases PDE font object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_font_release(
        HIG_PDE_FONT hFont
);
```

Arguments:

Name	Type	Description
hFont	HIG_PDE_FONT	PDE font object to release.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.8 HIG_PDE_FORM

Handle to the PDE form object. A form is an element that corresponds to an instance of XObject Form on a page (or other containing stream such as another XObject Form or annotation form). The context associated with this instance includes the actual stream that represents the XObject Form and the initial conditions of the graphics state. The latter consists of the transformation matrix, initial color values, and so forth. It is possible to have two Forms that refer to the same XObject Form. The forms will exist at different places on the same page, depending on the transformation matrix. They may also have different colors or line stroking parameters. In the case of a transparency group, the opacity is specified in the gstate.

Within a Form, each element has its own gstate (or is a container, place, or group object). These gstates are independent of the parent Form gstate. Form elements may have their own opacity.

Content may be obtained from a Form to edit the form's display list.

Members:

IG_PDE_form_create	Creates a new form from an existing object.
IG_PDE_form_clone	Creates a new form from an existing form object.
IG_PDE_form_get_content	Gets a PDE Content object for a form.
IG_PDE_form_set_content	Sets the form content.
IG_PDE_form_has_xgroup	Determines whether the XObject form has a Transparency XGroup.
IG_PDE_form_get_xgroup	Acquires the transparency group dictionary of the XObject form.
IG_PDE_form_set_xgroup	Sets the transparency group dictionary of the form XObject.

1.3.3.4.3.8.1 IG_PDE_form_create

Creates a new form from an existing object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_create (
        HIG_PDF_BASOBJ hXObject,
        HIG_PDF_BASOBJ hResources,
        LPAT_PDF_FIXEDMATRIX lpMatrix,
        LPHIG_PDE_FORM lphForm
);
```

Arguments:

Name	Type	Description
hXObject	HIG_PDF_BASOBJ	XObject from which a form is created.
hResources	HIG_PDF_BASOBJ	The hXObject's Resources dictionary. If you do not pass in a Resource object, subsequent calls to IG_PDF_page_get_content will fail (after the file is saved).
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX that holds the transformation matrix to use for the form.
lphForm	LPHIG_PDE_FORM	The newly created form object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose of the object.

1.3.3.4.3.8.2 IG_PDE_form_clone

Creates a new form from an existing form object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_clone (
        HIG_PDE_FORM hForm,
        LPHIG_PDE_FORM lphCloneForm
);
```

Arguments:

Name	Type	Description
hForm	HIG_PDE_FORM	Form object from which a new PDE Form is created.
lphCloneForm	LPHIG_PDE_FORM	The newly created form object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Creates a copy of the PDE Form, including the underlying objects.

1.3.3.4.3.8.3 IG_PDE_form_get_content

Gets a PDE Content object for a form.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_get_content(
        HIG_PDE_FORM hForm,
        LPHIG_PDE_CONTENT lphContent
);
```

Arguments:

Name	Type	Description
hForm	HIG_PDE_FORM	The form whose content is obtained.
lphContent	LPHIG_PDE_CONTENT	Form content object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.8.4 IG_PDE_form_set_content

Sets the form content.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_set_content(
        HIG_PDE_FORM hForm,
        HIG_PDE_CONTENT hContent
);
```

Arguments:

Name	Type	Description
hForm	HIG_PDE_FORM	The form whose content is set.
hContent	LPHIG_PDE_CONTENT	The new content for form.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.8.5 IG_PDE_form_has_xgroup

Determines whether the XObject form has a Transparency XGroup.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_has_xgroup(
        HIG_PDE_FORM hForm,
        LPAT_PDF_BOOL lpbHasXGroup
);
```

Arguments:

Name	Type	Description
hForm	HIG_PDE_FORM	The form object.
lpbHasXGroup	LPAT_PDF_BOOL	TRUE if the XObject form has a Transparency XGroup; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.8.6 IG_PDE_form_get_xgroup

Acquires the transparency group dictionary of the XObject form.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_get_xgroup(
        HIG_PDE_FORM hForm,
        LPHIG_PDE_XGROUP lphXGroup
);
```

Arguments:

Name	Type	Description
hForm	HIG_PDE_FORM	The form whose XGroup is obtained.
lphXGroup	LPHIG_PDE_XGROUP	Transparency group object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.8.7 IG_PDE_form_set_xgroup

Sets the transparency group dictionary of the form XObject.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_form_set_xgroup(
        HIG_PDE_FORM hForm,
        HIG_PDE_XGROUP hXGroup
);
```

Arguments:

Name	Type	Description
hForm	HIG_PDE_FORM	The form whose XGroup is set.
hXGroup	LPHIG_PDE_XGROUP	The transparency dictionary.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.9 HIG_PDE_GROUP

Handle to the PDE group object. A group is an in-memory representation of objects in Content. It has no state and is not represented in any way in a content stream (that is, Content).

When used in a Clip, this object is used to associate Text objects into a single clipping object.

Members:

IG_PDE_group_create	Creates a PDE Group object.
IG_PDE_group_get_content	Gets a PDE Content object for a group.
IG_PDE_group_set_content	Sets the group's content.

1.3.3.4.3.9.1 IG_PDE_group_create

Creates a PDE Group object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_group_create (
        LPHIG_PDE_GROUP lphGroup
);
```

Arguments:

Name	Type	Description
lphGroup	LPHIG_PDE_GROUP	The newly created group object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.9.2 IG_PDE_group_get_content

Gets a PDE Content object for a group.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_group_get_content(
        HIG_PDE_GROUP hGroup,
        LPHIG_PDE_CONTENT lphContent
);
```

Arguments:

Name	Type	Description
hGroup	HIG_PDE_GROUP	The group whose content is obtained.
lphContent	LPHIG_PDE_CONTENT	Group content object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.9.3 IG_PDE_group_set_content

Sets the group's content.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_group_set_content(
        HIG_PDE_GROUP hGroup,
        HIG_PDE_CONTENT hContent
);
```

Arguments:

Name	Type	Description
hGroup	HIG_PDE_GROUP	The group whose content is set.
hContent	HIG_PDE_CONTENT	The new content for group.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10 HIG_PDE_IMAGE

Handle to the PDE image object. An image is an element that contains an Image XObject or in-line image. You can associate data or a stream with an image.

Members:

IG_PDE_image_create	Creates an image object.
IG_PDE_image_is_data_encoded	Determines if image data is encoded or not.
IG_PDE_image_get_attrs	Gets the attributes for an image.
IG_PDE_image_get_color_mask	Gets the Mask entry from the image dictionary.
IG_PDE_image_get_colorspace	Gets the color space object for an image.
IG_PDE_image_get_data	Gets an image's data.
IG_PDE_image_get_data_length	Gets the length of data for an image.
IG_PDE_image_get_data_stream	Gets a data stream for an image.
IG_PDE_image_get_decode_array	Gets the decode array for an image.
IG_PDE_image_get_dictionary	Gets the dictionary for an image.
IG_PDE_image_get_filter_array	Gets the filter array for an image.
IG_PDE_image_get_matte_array	Gets the matte array for the image XObject.
IG_PDE_image_get_soft_mask	Gets the soft mask for an image.
IG_PDE_image_has_soft_mask	Checks whether the image has a soft mask.
IG_PDE_image_is_xobject	Determines if an image is an XObject image.
IG_PDE_image_set_color_mask	Sets the color space of the image.
IG_PDE_image_set_colorspace	Sets the Mask entry from the image dictionary.
IG_PDE_image_set_data	Sets data for an image.
IG_PDE_image_set_data_stream	Sets a data stream for an image.
IG_PDE_image_set_decode_array	Sets the decode array of an image.
IG_PDE_image_set_matte_array	Sets the matte array for the image XObject.
IG_PDE_image_set_soft_mask	Sets the soft mask.

1.3.3.4.3.10.1 IG_PDE_image_create

Creates an image object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_create(
        LPAT_PDE_IMAGEATTRS lpAttrs,
        LPAT_PDF_FIXEDMATRIX lpMatrix,
        LONG nFlag,
        HIG_PDE_COLORSPACE hColorSpace,
        LPAT_PDE_COLORVALUE lpColorValue,
        LPAT_PDE_FILTERARRAY lpFilters,
        HIG_PDF_STREAM hDataStream,
        LPBYTE lpData,
        LONG nDataLen,
        LPHIG_PDE_IMAGE lphImage
);
```

Arguments:

Name	Type	Description
lpAttrs	LPAT_PDE_IMAGEATTRS	Pointer to AT_PDE_IMAGEATTRS with attributes of the image.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX that holds the transformation matrix to use for the image.
nFlag	LONG	enumIGPDEImageDataFlags flags. If the AT_PDE_IMAGE_ENCODED_DATA flag is set, and the data is provided directly (not as a stream), then nDataLen must specify the length of data.
hColorSpace	HIG_PDE_COLORSPACE	Color space of the image. When the image is an imagemask, hColorSpace is the color space of the lpColorValue argument.
lpColorValue	LPAT_PDE_COLORVALUE	Pointer to AT_PDE_COLORVALUE structure. If the image is an image mask, lpColorValue must be provided.
lpFilters	LPAT_PDE_FILTERARRAY	Pointer to AT_PDE_FILTERARRAY structure that specifies which filters to use in encoding the contents; may be NULL. Filters will be used to encode the data in the order in which they are specified in the array.
hDataStream	HIG_PDF_STREAM	Stream holding the image data.
lpData	LPBYTE	Image data. If hDataStream is non-NULL, data is ignored. If there is a great deal of data, as for a large image, it is recommended you use the hDataStream parameter for the image data.
nDataLen	LONG	Encoded length of lpData, in bytes.
lphImage	LPHIG_PDE_IMAGE	The image object.

Return Value:

Error count.

Supported Raster Image Formats:

See Section 4.8 of the PDF Reference for information on image types supported by the PDF format.

Remarks:

The image data may be specified as a stream or as a buffer. If hDataStream is non-NULL, lpData is ignored.

See IG_PDE_image_set_data_stream for information on handling the stream.

The caller must dispose of hDataStream after calling this function.

Call IG_PDE_element_release to dispose the created image object when finished with it.

1.3.3.4.3.10.2 IG_PDE_image_is_data_encoded

Determines if image data is encoded or not. Used only for inline images; not relevant to XObject images.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_is_data_encoded(
        HIG_PDE_IMAGE hImage,
        LPAT_PDF_BOOL lpbIsEncoded,
        LPDWORD lpnEncodedLen
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image to examine.
lpbIsEncoded	LPAT_PDF_BOOL	TRUE if IG_PDE_image_get_data returns encoded data; FALSE otherwise. Returns FALSE for XObject images.
lpnEncodedLen	LPDWORD	Length of the encoded data-if the data is encoded, that is, if lpbIsEncoded returns TRUE.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

lpbIsEncoded always returns FALSE for XObject images; XObject image data can be obtained from IG_PDE_image_get_data or IG_PDE_image_get_data_stream, either encoded or decoded.

Only if IG_PDE_image_create is used to explicitly create a new image using encoded data does lpbIsEncoded returns TRUE.

1.3.3.4.3.10.3 IG_PDE_image_get_attrs

Gets the attributes for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_attrs (
        HIG_PDE_IMAGE hImage,
        LPAT_PDE_IMAGEATTRS lpAttrs
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose attributes are obtained.
lpAttrs	LPAT_PDE_IMAGEATTRS	Pointer to AT_PDE_IMAGEATTRS structure with attributes of the image.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.4 IG_PDE_image_get_color_mask

Use this function to obtain the Mask entry from the image dictionary.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_color_mask(
        HIG_PDE_IMAGE hImage,
        LPLONG lpMask,
        LPUINT lpnLen
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	The image object whose color mask is obtained.
lpMask	LPLONG	A pointer to the array of LONG values to fill with color mask values. lpMask must contain enough values to hold the entire color mask array.
lpnLen	LPUINT	The number of color mask elements obtained by the method - size of lpMask array.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The mask entry is an array specifying a range of colors to be masked out. Samples in the image that fall within this range are not painted, allowing the existing background to show through. The effect is similar to that of the video technique known as chroma-key.

The value of each Mask entry is an array of 2n integers, [min1 max1 ... minn maxn], where n is the number of color components in the image's color space. Each integer must be in the range 0 to (2^BitsPerComponent - 1), representing color values before decoding with the Decode array. An image sample is masked (not painted) if all of its color components before decoding, c1...cn, fall within the specified ranges.

1.3.3.4.3.10.5 IG_PDE_image_get_colorspace

Gets the color space object for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_colorspace (
        HIG_PDE_IMAGE hImage,
        LPHIG_PDE_COLORSPACE lphColorSpace
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose color space is obtained.
lphColorSpace	LPHIG_PDE_COLORSPACE	Color space for hImage. Returns NULL if hImage is an image mask.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.6 IG_PDE_image_get_data

Gets an image's data.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_data (
        HIG_PDE_IMAGE hImage,
        LONG nFlags,
        LPBYTE lpData
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose data is obtained.
nFlags	LONG	Unused, must be 0.
lpData	LPBYTE	Image data. If the data is decoded, lpData must be large enough to contain the number of bytes specified in the AT_PDE_IMAGEATTRS structure obtained by IG_PDE_image_get_attrs. If the data is encoded, lpData must be large enough to contain the number of bytes in the lpnEncodedLen parameter obtained by IG_PDE_image_is_data_encoded.

Remarks:

If the image is a XObject image, data is always returned as decoded data.

See the note about inline images under IG_PDE_image_is_data_encoded.

Supported Raster Image Formats:

This function does not process image pixels.

Return Value:

Error count.

1.3.3.4.3.10.7 IG_PDE_image_get_data_length

Gets the length of data for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_data_length (
        HIG_PDE_IMAGE hImage,
        LPLONG lpnLength
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose data length is obtained.
lpnLength	LPLONG	Number of bytes of image data, specified by the width, height, bits per component, and color space of the image.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.8 IG_PDE_image_get_data_stream

Gets a data stream for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_data_stream (
        HIG_PDE_IMAGE hImage,
        LONG nFlags,
        LPHIG_PDF_STREAM lphStream
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose data stream is obtained.
nFlags	LONG	enumIGPDEImageDataFlags flags. If the AT_PDE_IMAGE_ENCODED_DATA flag is set, data is returned in encoded form. Otherwise, data is decoded.
lphStream	LPHIG_PDF_STREAM	Stream for hImage.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

May only be called for XObject images.

The caller must dispose of the returned stream by calling IG_PDF_stream_close.

1.3.3.4.3.10.9 IG_PDE_image_get_decode_array

Gets the decode array for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_decode_array (
        HIG_PDE_IMAGE hImage,
        LPAT_PDF_FIXED lpDecode,
        LONG nDecodeSize,
        LPLONG lpnLength
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	The image whose decode array is obtained.
lpDecode	LPAT_PDF_FIXED	Pointer to the decode array. If NULL, the number of decode elements required is returned via lpnLength.
nDecodeSize	LONG	Size of lpDecode in bytes.
lpnLength	LPLONG	Number of elements in the decode array.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.10 IG_PDE_image_get_dictionary

Gets the dictionary for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_dictionary(
        HIG_PDE_IMAGE hImage,
        LPHIG_PDF_BASOBJ lphDictionary
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	An image object.
lphDictionary	LPHIG_PDF_BASOBJ	Dictionary for hImage.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.11 IG_PDE_image_get_filter_array

Gets the filter array for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_filter_array (
        HIG_PDE_IMAGE hImage,
        LPAT_PDE_FILTERARRAY lpFilters
        LPLONG lpnLength
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose filter array is obtained.
lpFilters	LPAT_PDE_FILTERARRAY	Pointer to AT_PDE_FILTERARRAY structure to fill with the current filter array for the image. lpFilters must be large enough to contain all of the elements. May be NULL to obtain the number of filter elements via lpnLength.
lpnLength	LPLONG	Number of filter elements.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.12 IG_PDE_image_get_matte_array

Gets the matte array for the image XObject.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_matte_array (
        HIG_PDE_IMAGE hImage,
        LPAT_PDF_FIXED lpMatte,
        LONG nValuesCount,
        LPLONG lpnLength
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	The image XObject.
lpMatte	LPAT_PDF_FIXED	An array of values.
nValuesCount	LONG	The number of values in lpMatte.
lpnLength	LPLONG	Number of values copied.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.13 IG_PDE_image_get_soft_mask

Gets the soft mask for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_get_soft_mask (
        HIG_PDE_IMAGE hImage,
        LPHIG_PDE_IMAGE lphSoftMask
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	The image object.
lphSoftMask	LPHIG_PDE_IMAGE	The soft mask for image.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Use IG_PDE_element_release to dispose of the object when it is no longer referenced.

1.3.3.4.3.10.14 IG_PDE_image_has_soft_mask

Checks whether the image has a soft mask.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_has_soft_mask (
        HIG_PDE_IMAGE hImage,
        LPAT_PDF_BOOL lpbHasSoftMask
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	The image object.
lpbHasSoftMask	LPAT_PDF_BOOL	TRUE if the soft mask exists; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.15 IG_PDE_image_is_xobject

Determines if an image is an XObject image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_is_xobject (
        HIG_PDE_IMAGE hImage,
        LPAT_PDF_BOOL lpbIsXObject
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	The image object.
lpbIsXObject	LPAT_PDF_BOOL	TRUE if the image is an XObject image; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.16 IG_PDE_image_set_color_mask

Use this function to set the Mask entry from the image dictionary.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_color_mask(
        HIG_PDE_IMAGE hImage,
        LPLONG lpMask,
        UINT nLen
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	The image object whose color mask is set.
lpMask	LPLONG	A pointer to the array of LONG values containing the color mask values.
nLen	UINT	The number of color mask elements in lpMask - size of lpMask array.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Example:

```
//24-bit rgb color mask with 1 mask entry
LONG lpMask[] = {250, 255, 250, 255, 250, 255};
int iMaskLen=6;
IG_PDE_image_set_color_mask(hElement,lpMask,iMaskLen);
```

Remarks:

The mask entry is an array specifying a range of colors to be masked out. Samples in the image that fall within this range are not painted, allowing the existing background to show through. The effect is similar to that of the video technique known as chroma-key.

The value of each Mask entry is an array of 2n integers, [min1 max1 ... minn maxn], where n is the number of color components in the image's color space. Each integer must be in the range 0 to (2^BitsPerComponent - 1), representing color values before decoding with the Decode array. An image sample is masked (not painted) if all of its color components before decoding, c1...cn, fall within the specified ranges.

1.3.3.4.3.10.17 IG_PDE_image_set_colorspace

Sets the color space of the image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_colorspace (
        HIG_PDE_IMAGE hImage,
        HIG_PDE_COLORSPACE hColorSpace
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose color space is set.
hColorSpace	HIG_PDE_COLORSPACE	PDE ColorSpace object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.18 IG_PDE_image_set_data

Sets data for an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_data (
        HIG_PDE_IMAGE hImage,
        LONG nFlags,
        LPBYTE lpData,
        LONG nLength
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose data is set.
nFlags	LONG	A set of enumIGPDEImageDataFlags flags. If AT_PDE_IMAGE_ENCODED_DATA is set, the data must be encoded for the current filters, and nLength is the length of the encoded data. If the AT_PDE_IMAGE_ENCODED_DATA flag is not set, data is not encoded and nLength is the size of the decoded data.
lpData	LPBYTE	Image data.
nLength	LONG	Length of data.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.19 IG_PDE_image_set_data_stream

Sets a data stream for an image; can only be used for XObject images.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_data_stream (
        HIG_PDE_IMAGE hImage,
        LONG nFlags,
        LPAT_PDE_FILTERARRAY lpFilters,
        HIG_PDF_STREAM hDataStream
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose data stream is set.
nFlags	LONG	enumIGPDEImageDataFlags flags. If the AT_PDE_IMAGE_ENCODED_DATA flag is set, the stream must be encoded.
lpFilters	LPAT_PDE_FILTERARRAY	Pointer to AT_PDE_FILTERARRAY structure. If not NULL, is used to build the objects for the Filter, DecodeParms, and EncodeParms objects. If lpFilters is NULL, the existing Filter and DecodeParms are used. EncodeParms is set to DecodeParms if it exists.
hDataStream	HIG_PDF_STREAM	Stream for the image data.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.20 IG_PDE_image_set_decode_array

Sets the decode array of an image.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_decode_array (
        HIG_PDE_IMAGE hImage,
        LPAT_PDF_FIXED lpDecode,
        LONG nDecodeSize
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	Image whose decode array is set.
lpDecode	LPAT_PDF_FIXED	Pointer to the decode array.
nDecodeSize	LONG	Size of decode array in bytes.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Normally, the decode array is accessed through the decode field in the AT_PDE_IMAGEATTRS structure. However, this function defines a decode array to handle images with a color space that has more than 4 components.

1.3.3.4.3.10.21 IG_PDE_image_set_matte_array

Sets the matte array for the image XObject.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_matte_array (
        HIG_PDE_IMAGE hImage,
        LPAT_PDF_FIXED lpMatte,
        LONG nValuesCount
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	The image XObject.
lpMatte	LPAT_PDF_FIXED	An array of values.
nValuesCount	LONG	The number of values in lpMatte.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.10.22 IG_PDE_image_set_soft_mask

Sets the soft mask.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_image_set_soft_mask (
        HIG_PDE_IMAGE hImage,
        HIG_PDE_IMAGE hSoftMask
);
```

Arguments:

Name	Type	Description
hImage	HIG_PDE_IMAGE	The image XObject.
hSoftMask	HIG_PDE_IMAGE	The soft mask.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.11 HIG_PDE_PATH

Handle to the PDE path object. A path is an element that contains a path. Path objects can be stroked, filled, and/or serve as a clipping path.

Members:

IG_PDE_path_create	Creates an empty path element.
IG_PDE_path_add_segment	Adds a segment to a path.
IG_PDE_path_get_data	Gets the size of the path data and, optionally, the path data.
IG_PDE_path_set_data	Sets new path data for a path element.
IG_PDE_path_get_paint_op	Gets the fill and stroke attributes of a path.
IG_PDE_path_set_paint_op	Sets the fill and stroke attributes of a path.

1.3.3.4.3.11.1 IG_PDE_path_create

Creates an empty path element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_create(
        LPHIG_PDE_PATH lphPath
);
```

Arguments:

Name	Type	Description
lphPath	LPHIG_PDE_PATH	Newly created empty path element.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose of the created path object when finished with it.

1.3.3.4.3.11.2 IG_PDE_path_add_segment

Adds a segment to a path.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_add_segment(
        HIG_PDE_PATH hPath,
        LONG nSegType,
        AT_PDF_FIXED x1,
        AT_PDF_FIXED y1,
        AT_PDF_FIXED x2,
        AT_PDF_FIXED y2,
        AT_PDF_FIXED x3,
        AT_PDF_FIXED y3
);
```

Arguments:

Name	Type	Description
hPath	HIG_PDE_PATH	The path to which a segment is added.
nSegType	LONG	A enumIGPDEPathElementType value indicating the type of path to add.
x1	AT_PDF_FIXED	x-coordinate of first point.
y1	AT_PDF_FIXED	y-coordinate of first point.
x2	AT_PDF_FIXED	x-coordinate of second point.
y2	AT_PDF_FIXED	y-coordinate of second point.
x3	AT_PDF_FIXED	x-coordinate of third point.
y3	AT_PDF_FIXED	y-coordinate of third point.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The number of AT_PDF_FIXED values used depends upon nSegType:

- IG_PDE_MOVE_TO: x1, y1
- IG_PDE_LINE_TO: x1, y1
- IG_PDE_CURVE_TO: x1, y1, x2, y2, x3, y3
- IG_PDE_CURVE_TO_V: x1, y1, x2, y2
- IG_PDE_CURVE_TO_Y: x1, y1, x2, y2
- IG_PDE_RECT: x1, y1, x2 (width), y2 (height)
- IG_PDE_CLOSE_PATH: None

1.3.3.4.3.11.3 IG_PDE_path_get_data

Gets the size of the path data and, optionally, the path data.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_get_data(
        HIG_PDE_PATH hPath,
        LPLONG lpData,
        LONG nDataSize,
        LPLONG lpnLenght
);
```

Arguments:

Name	Type	Description
hPath	HIG_PDE_PATH	The path whose data is obtained.
lpData	LPLONG	Pointer to path data. If lpData is non-NULL, it contains a variable-sized array of path operators and operands. The format is a 32-bit operator followed by 0 to 3 AT_PDF_FIXEDPOINT values, depending on the operator. Opcodes are codes for moveto, lineto, curveto, rect, or closepath operators; operands are AT_PDF_FIXEDPOINT values. If data is NULL, the number of bytes required for data is returned in lpnLenght. Returns "raw" path data. If you want the points in page coordinates, concatenate the path data points with the PDE Element matrix obtained from IG_PDE_element_get_matrix.
nDataSize	LONG	Specifies the size of the buffer provided in data. If it is less than the length of the path data, the method copies datasize bytes.
lpnLength	LPLONG	Length of data of hPath.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.11.4 IG_PDE_path_set_data

Sets new path data for a path element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_set_data(
        HIG_PDE_PATH hPath,
        LPLONG lpData,
        LONG nDataSize
);
```

Arguments:

Name	Type	Description
hPath	HIG_PDE_PATH	The path whose data is set.
lpData	LPLONG	Pointer to path data. It is a variable-sized array of path operators and operands. The format is a 32-bit operator followed by 0 to 3 AT_PDF_FIXEDPOINT values, depending on the operator. Operators are codes for moveto, lineto, curveto, rect, or closepath operators and must be one of enumIGPDEPathElementType. Operands are AT_PDF_FIXEDPOINT values. The data is copied into hPath object.
nDataSize	LONG	Size of the new path data, in bytes.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.11.5 IG_PDE_path_get_paint_op

Gets the fill and stroke attributes of a path.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_get_paint_op(
        HIG_PDE_PATH hPath,
        LPLONG lpnPaintOpAttrs
);
```

Arguments:

Name	Type	Description
hPath	HIG_PDE_PATH	The path whose fill and stroke attributes are obtained.
lpnPaintOpAttrs	LPLONG	A set of enumIGPDEPathOpFlags flags describing fill and stroke attributes.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.11.6 IG_PDE_path_set_paint_op

Sets the fill and stroke attributes of a path.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_path_set_paint_op(
        HIG_PDE_PATH hPath,
        LONG nPaintOpAttrs
);
```

Arguments:

Name	Type	Description
hPath	HIG_PDE_PATH	The path whose fill and stroke attributes are set.
nPaintOpAttrs	LONG	The operation to set; must be one of enumIGPDEPathOpFlags.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.12 HIG_PDE_PLACE

Handle to the PDE place object. A place is an element that marks a place on a page in a PDF file. In a PDF file, a place is represented by the MP or DP Marked Content operators.

Marked content is useful for adding structure information to a PDF file. For instance, a drawing program may want to mark a point with information, such as the start of a path of a certain type. Marked content provides a way to retain this information in the PDF file. A DP operator functions the same as the MP operator and, in addition, allows a property list dictionary to be associated with a place.

Members:

IG_PDE_place_create	Creates a place object.
IG_PDE_place_get_dictionary	Gets the Marked Content dictionary for hPlace.
IG_PDE_place_set_dictionary	Sets the Marked Content dictionary for hPlace.
IG_PDE_place_get_mctag	Gets the Marked Content tag for a hPlace.
IG_PDE_place_set_mctag	Sets the Marked Content tag for a hPlace.

1.3.3.4.3.12.1 IG_PDE_place_create

Creates a place object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_create(
        HIG_PDF_ATOM mcTag,
        HIG_PDF_BASOBJ mcDict,
        AT_PDF_BOOL bIsInline
        LPHIG_PDE_PLACE lphPlace
);
```

Arguments:

Name	Type	Description
mcTag	HIG_PDF_ATOM	Tag name for the place. Must not contain any white space characters (for example, spaces or tabs).
mcDict	HIG_PDF_BASOBJ	Optional Marked Content dictionary associated with the place.
bIsInline	AT_PDF_BOOL	If TRUE, place is emitted into the page content stream inline.
lphPlace	LPHIG_PDE_PLACE	The place object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose the created place object when finished with it.

1.3.3.4.3.12.2 IG_PDE_place_get_dictionary

Gets the Marked Content dictionary for hPlace.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_get_dictionary(
        HIG_PDE_PLACE hPlace,
        LPHIG_PDF_BASOBJ lpmcDict,
        LPAT_PDF_BOOL lpbIsInline,
        LPAT_PDF_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hPlace	HIG_PDE_PLACE	The place whose Marked Content dictionary is obtained.
lpmcDict	LPHIG_PDF_BASOBJ	Pointer to the Marked Content dictionary; may be NULL.
lpbIsInline	LPAT_PDF_BOOL	If TRUE, the Marked Content dictionary is inline; may be NULL.
lpbResult	LPAT_PDF_BOOL	TRUE if dictionary is obtained; FALSE if no dictionary is present.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.12.3 IG_PDE_place_set_dictionary

Sets the Marked Content dictionary for hPlace.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_set_dictionary(
        HIG_PDE_PLACE hPlace,
        HIG_PDF_BASOBJ mcDict,
        AT_PDF_BOOL bIsInline
);
```

Arguments:

Name	Type	Description
hPlace	HIG_PDE_PLACE	The place whose Marked Content dictionary is set.
lpmcDict	HIG_PDF_BASOBJ	Marked Content dictionary for hPlace.
lpbIsInline	AT_PDF_BOOL	If TRUE, the dictionary is emitted inline.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.12.4 IG_PDE_place_get_mctag

Gets the Marked Content tag for a hPlace.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_get_mctag(
        HIG_PDE_PLACE hPlace,
        LPHIG_PDF_ATOM lpmcTag
);
```

Arguments:

Name	Type	Description
hPlace	HIG_PDE_PLACE	The place whose Marked Content tag is obtained.
lpmcTag	LPHIG_PDF_ATOM	Tag for hPlace.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.12.5 IG_PDE_place_set_mctag

Sets the Marked Content tag for a hPlace.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_place_set_mctag(
        HIG_PDE_PLACE hPlace,
        HIG_PDF_ATOM mcTag
);
```

Arguments:

Name	Type	Description
hPlace	HIG_PDE_PLACE	The place whose Marked Content tag is set.
mcTag	HIG_PDF_ATOM	The tag for hPlace.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.13 HIG_PDE_POSTSCRIPT

Handle to the PDE PostScript object. PostScript - an element representing in-line or XObject pass-through PostScript object. XObject PostScripts are listed in page XObject resources.

Members:

IG_PDE_postscript_create	Creates a PDE PostScript object.
IG_PDE_postscript_get_attrs	Gets hPostScript attributes.
IG_PDE_postscript_get_data	Gets all or part of the image data.
IG_PDE_postscript_set_data	Sets the data for hPostScript.
IG_PDE_postscript_get_data_stream	Gets a stream for the data.
IG_PDE_postscript_set_data_stream	Sets a stream for the data.

1.3.3.4.3.13.1 IG_PDE_postscript_create

Creates a PDE PostScript object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_create(
        LPAT_PDE_PSATTRS lpAttrs,
        HIG_PDF_STREAM hDataStream,
        LPBYTE lpData,
        LONG nDataSize
        LPHIG_PDE_POSTSCRIPT lphPostScript
);
```

Arguments:

Name	Type	Description
lpAttrs	LPAT_PDE_PSATTRS	Pointer to AT_PDE_PSATTRS attributes data structure.
hDataStream	HIG_PDF_STREAM	Data stream. May be NULL.
lpData	LPBYTE	Data. May be NULL.
nDataSize	LONG	Number of bytes of data.
lphPostScript	LPHIG_PDE_POSTSCRIPT	The postscript object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

lpData and hDataStream may be NULL. If so, use IG_PDE_postscript_set_data and IG_PDE_postscript_set_data_stream to attach data to the object. If hDataStream is non-NULL, then data will be ignored.

If data is non-NULL and hDataStream is NULL, the data must contain nDataSize number of bytes as specified in the AT_PDE_PSATTRS.

Call IG_PDE_element_release to dispose of the created object when finished with it.

1.3.3.4.3.13.2 IG_PDE_postscript_get_attrs

Gets hPostScript attributes.

Declaration:

```
AT_ERRCOUNT ACCUAPIIG_PDE_postscript_get_attrs(
        HIG_PDE_POSTSCRIPT hPostScript,
        LPAT_PDE_PSATTRS lpAttrs
);
```

Arguments:

Name	Type	Description
hPostScript	HIG_PDE_POSTSCRIPT	PDE postscript object.
lpAttrs	LPAT_PDE_PSATTRS	Pointer to AT_PDE_PSATTRS data structure containing the attributes information.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.13.3 IG_PDE_postscript_get_data

Gets all or part of the image data.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_get_data (
        HIG_PDE_POSTSCRIPT hPostScript,
        LPBYTE lpBuffer,
        LONG nBufferSize,
        LONG nOffset,
        LPLONG lpnBytesWritten
);
```

Arguments:

Name	Type	Description
hPostScript	HIG_PDE_POSTSCRIPT	PDE postscript object.
lpBuffer	LPBYTE	Receives the data.
nBufferSize	LONG	Size of the buffer.
nOffset	LONG	Offset into the source data at which to start filling buffer.
lpnBytesWritten	LPLONG	The number of bytes written into the buffer. If it is less than nBufferSize, then there is no more data.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.13.4 IG_PDE_postscript_set_data

Sets the data for hPostScript.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_set_data (
        HIG_PDE_POSTSCRIPT hPostScript,
        LPBYTE lpBuffer,
        LONG nBufferSize
);
```

Arguments:

Name	Type	Description
hPostScript	HIG_PDE_POSTSCRIPT	PDE postscript object.
lpBuffer	LPBYTE	Contains the data.
nBufferSize	LONG	Length of the data in bytes.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.13.5 IG_PDE_postscript_get_data_stream

Gets a stream for the data.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_get_data_stream (
        HIG_PDE_POSTSCRIPT hPostScript,
        LPHIG_PDF_STREAM lphStream
);
```

Arguments:

Name	Type	Description
hPostScript	HIG_PDE_POSTSCRIPT	PDE postscript object.
lphStream	LPHIG_PDF_STREAM	Stream for hPostScript.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The data in the stream is decoded (no filters).

The caller must dispose of the returned stream by calling IG_PDF_stream_close.

1.3.3.4.3.13.6 IG_PDE_postscript_set_data_stream

Sets a stream for the data; the data must be un-encoded (no filters).

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_postscript_set_data_stream (
        HIG_PDE_POSTSCRIPT hPostScript,
        HIG_PDF_STREAM hStream
);
```

Arguments:

Name	Type	Description
hPostScript	HIG_PDE_POSTSCRIPT	PDE postscript object.
hStream	HIG_PDF_STREAM	Stream for the data.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.14 HIG_PDE_SOFTMASK

Handle to the PDE soft mask object. A soft mask is an object for creating and manipulating a soft mask in a PDF file.

Members:

IG_PDE_softmask_create	Creates a new soft mask object.
IG_PDE_softmask_create_from_name	Create a new soft mask from a name.
IG_PDE_softmask_get_form	Acquires the form that defines the soft mask.
IG_PDE_softmask_set_form	Sets the form that defines the soft mask.
IG_PDE_softmask_get_backdrop_color	Gets the array of color values of the backdrop color.
IG_PDE_softmask_set_backdrop_color	Sets the backdrop color values.
IG_PDE_softmask_get_name	Gets the soft mask name.

1.3.3.4.3.14.1 IG_PDE_softmask_create

Creates a new soft mask object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_create(
        HIG_PDF_DOC hDoc,
        LONG nType,
        HIG_PDE_FORM hForm,
        LPHIG_PDE_SOFTMASK lphSoftMask
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The container document.
nType	LONG	Specifies how the mask is to be computed. One of the enumIGPDESoftMaskCreateFlags.
hForm	HIG_PDE_FORM	The form XObject that defines the soft mask. It is the source of the mask values and the color space in which the composite computation is to be done.
lphSoftMask	LPHIG_PDE_SOFTMASK	The newly created object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose the created object when finished with it.

1.3.3.4.3.14.2 IG_PDE_softmask_create_from_name

Create a new soft mask from a name.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_create_from_name(
        HIG_PDF_ATOM hName,
        LPHIG_PDE_SOFTMASK lphSoftMask
);
```

Arguments:

Name	Type	Description
hName	HIG_PDF_ATOM	The new name for the soft mask. Currently, the only valid name is None.
lphSoftMask	LPHIG_PDE_SOFTMASK	The newly created object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose the created object when finished with it.

1.3.3.4.3.14.3 IG_PDE_softmask_get_form

Acquires the form that defines the soft mask.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_get_form(
        HIG_PDE_SOFTMASK hSoftMask,
        LPAT_PDF_FIXEDMATRIX lpMatrix,
        LPHIG_PDE_FORM lphForm
);
```

Arguments:

Name	Type	Description
hSoftMask	HIG_PDE_SOFTMASK	The soft mask object.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Matrix defining the transformation from coordinate space to user space.
lphForm	LPHIG_PDE_FORM	The XObject form of the soft mask.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.14.4 IG_PDE_softmask_set_form

Sets the form that defines the soft mask.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_set_form(
        HIG_PDE_SOFTMASK hSoftMask,
        HIG_PDE_FORM hForm
);
```

Arguments:

Name	Type	Description
hSoftMask	HIG_PDE_SOFTMASK	The soft mask object.
hForm	HIG_PDE_FORM	The form XObject.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.14.5 IG_PDE_softmask_get_backdrop_color

Gets the array of color values of the backdrop color.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_get_backdrop_color (
        HIG_PDE_SOFTMASK hSoftMask,
        LPAT_PDF_FIXED lpColorValues,
        LONG nColorValuesLen,
        LPLONG lpnCount
);
```

Arguments:

Name	Type	Description
hSoftMask	HIG_PDE_SOFTMASK	The soft mask object.
lpColorValues	LPAT_PDF_FIXED	Pointer to an array of color values. If NULL, the number of color values is returned in lpnCount.
nColorValuesLen	LONG	Length of the array lpColorValues.
lpnCount	LPLONG	Number of values copied.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Given a pointer to an array and the length of the array, copies the color values to that array and returns the number of values copied. If the pointer to the array is NULL, the number of color values is returned in lpnCount.

1.3.3.4.3.14.6 IG_PDE_softmask_set_backdrop_color

Sets the backdrop color values.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_set_backdrop_color (
        HIG_PDE_SOFTMASK hSoftMask,
        LPAT_PDF_FIXED lpColorValues,
        LONG nColorValuesLen
);
```

Arguments:

Name	**Type**	**Description**
hSoftMask	HIG_PDE_SOFTMASK	The soft mask object.
lpColorValues	LPAT_PDF_FIXED	Pointer to an array of color values.
nColorValuesLen	LONG	The number of values pointed to by lpColorValues.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.14.7 IG_PDE_softmask_get_name

Gets the soft mask name.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_softmask_get_name (
        HIG_PDE_SOFTMASK hSoftMask,
        LPHIG_PDF_ATOM lphName
);
```

Arguments:

Name	Type	Description
hSoftMask	HIG_PDE_SOFTMASK	The soft mask object.
lphName	LPHIG_PDF_ATOM	Soft mask name if it is a name; IG_PDF_ATOM_NULL otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.15 HIG_PDE_SHADING

Handle to the PDE shading object. Shading - an element that represents smooth shading.

Members:

IG_PDE_shading_create	Creates a smooth shading object.
IG_PDE_shading_get_dictionary	Gets the dictionary for a shading.

1.3.3.4.3.15.1 IG_PDE_shading_create

Creates a smooth shading object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_shading_create (
        HIG_PDF_BASOBJ hDictionary,
        LPAT_PDF_FIXEDMATRIX lpMatrix,
        LPHIG_PDE_SHADING lphShading
);
```

Arguments:

Name	Type	Description
hDictionary	HIG_PDF_BASOBJ	The shading dictionary.
lpMatrix	LPAT_PDF_FIXEDMATRIX	The location and transformation matrix of the shading object.
lphShading	LPHIG_PDE_SHADING	A smooth shading object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose of the created object when finished with it.

1.3.3.4.3.15.2 IG_PDE_shading_get_dictionary

Gets the dictionary for a shading.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_shading_get_dictionary(
        HIG_PDE_SHADING hShading,
        LPHIG_PDF_BASOBJ lphDictionary
);
```

Arguments:

Name	Type	Description
hShading	HIG_PDE_SHADING	A shading object.
lphDictionary	LPHIG_PDF_BASOBJ	Dictionary for hShading.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16 HIG_PDE_TEXT

Handle to the PDE text object. Text - an element representing text. It is a container for text as show strings or as individual characters. Each sub-element may have different graphics state properties. However, the same clip applies to all sub-elements of a Text. Also, the char path of a Text can be used to represent a clip.

Members:

IG_PDE_text_create	Creates an empty text object.
IG_PDE_text_add	Adds a character or a text run to a PDE Text object.
IG_PDE_text_add_item	Adds a text item to a text element at a given index position.
IG_PDE_text_get_advance	Gets the advance width of a character or a text element.
IG_PDE_text_get_bbox	Gets the bounding box of a character or a text run.
IG_PDE_text_get_font	Gets the font for a text character or element.
IG_PDE_text_get_gstate	Gets the graphics state of a character or a text run.
IG_PDE_text_get_item	Obtains a text item from a text element at a given index position.
IG_PDE_text_get_matrix	Returns the matrix of a character or a text element.
IG_PDE_text_get_byte_count	Gets the number of bytes occupied by the character code or text run.
IG_PDE_text_get_char_count	Gets the number of characters in a text object.
IG_PDE_text_get_runs_count	Gets the number of text runs (show strings) in a text object.
IG_PDE_text_get_quad	Gets the quad bounding the specified text run or character.
IG_PDE_text_get_run_for_char	Gets the index of the text run that contains the nth character in a text object.
IG_PDE_text_get_state	Gets the text state of a character or a text element.
IG_PDE_text_get_stroke_matrix	Gets the stroke matrix of a character or a text run.
IG_PDE_text_get_text	Gets the text for a text run or character.
IG_PDE_text_is_at_point	Tests whether a point is on specified text.
IG_PDE_text_is_at_rect	Tests whether any part of a rectangle is on the specified text.
IG_PDE_text_remove	Removes characters or text runs from a text object.
IG_PDE_text_remove_items	Removes contiguous text items from a text element starting at a given index position.
IG_PDE_text_replace_chars	Replaces characters in a text object.
IG_PDE_text_run_get_char_offset	Gets the character offset of the first character of the specified text run.
IG_PDE_text_run_get_char_count	Gets the number of characters in a text run.
IG_PDE_text_run_set_font	Sets the font of a text run.
IG_PDE_text_run_set_gstate	Sets the graphics state of a text run.
IG_PDE_text_run_set_matrix	Sets the matrix of a text run.
IG_PDE_text_run_set_state	Sets the text state of a text run.
IG_PDE_text_run_set_stroke_matrix	Sets the stroke matrix of a text run.
IG_PDE_text_split_run_at	Splits a text run into two text runs.

1.3.3.4.3.16.1 IG_PDE_text_create

Creates an empty text object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_create(
        LPHIG_PDE_TEXT lphText
);
```

Arguments:

Name	Type	Description
lphText	LPHIG_PDE_TEXT	An empty text object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose the created object when finished with it.

1.3.3.4.3.16.2 IG_PDE_text_add

Adds a character or a text run to a PDE Text object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_add(
        HIG_PDE_TEXT hText,
        UINT nFlag,
        UINT nIndex,
        LPBYTE lpText,
        UINT nTextLen,
        HIG_PDE_FONT hFont,
        LPAT_PDE_GRAPHICSTATE lpGstate,
        LPAT_PDE_TEXTSTATE lpTstate,
        LPAT_PDF_FIXEDMATRIX lpTextMatrix,
        LPAT_PDF_FIXEDMATRIX lpStrokeMatrix
);
```

Arguments:

Name	Type	Description
hText	LPHIG_PDE_TEXT	Text object to which a character or text run is added.
nFlag	UINT	enumIGPDETextFlags flag that specifies what kind of text to add. Must be either: • IG_PDE_TEXT_CHAR - for a text character • IG_PDE_TEXT_RUN - for a text run.
nIndex	UINT	Index after which to add character or text run.
lpText	LPBYTE	Pointer to the characters to add. Passing NULL for text can invalidate the text object but will not raise an error. Callers must not pass NULL for this parameter.
nTextLen	UINT	Length of the text, in bytes.
hFont	HIG_PDE_FONT	Font for the element.
lpGstate	LPAT_PDE_GRAPHICSTATE	Pointer to AT_PDE_GRAPHICSTATE structure with the graphics state for the element.
lpTstate	LPAT_PDE_TEXTSTATE	Pointer to AT_PDE_TEXTSTATE structure with text state for the element.
lpTextMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX that holds the matrix for the element.
lpStrokeMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX that holds the matrix for the line width when stroking text. May be NULL.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.3 IG_PDE_text_add_item

Adds a text item to a text element at a given index position.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_add_item(
        HIG_PDE_TEXT hText,
        UINT nIndex,
        HIG_PDE_TEXTITEM hTextItem
);
```

Arguments:

Name	**Type**	**Description**
hText	HIG_PDE_TEXT	Text object to which an item is added.
nIndex	UINT	Index of the text item in hText.
hTextItem	HIG_PDE_TEXTITEM	The text item to add.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.4 IG_PDE_text_get_advance

Gets the advance width of a character or a text element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_advance(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPAT_PDF_FIXEDPOINT lpAdvanceWidth
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose advance width is found.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run In addition, set the IG_PDE_TEXT_PAGE_SPACE flag to obtain the advance width in user space. If it is not set, the advance width is in character space. If this flag is not set, this function returns a value that is independent of any sizes, matrices, or scaling, simply adding up the font's raw glyph widths, supplemented only by nonscaled character and word spacing.
nIndex	UINT	Index of the character or text run in hText.
lpAdvanceWidth	LPAT_PDF_FIXEDPOINT	Pointer to AT_PDF_FIXEDPOINT value indicating the advance width.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Advance width is returned in either character space or user space. The advance width is the amount by which the current point advances when the character is drawn.

Advance width may be horizontal or vertical, depending on the writing style. Thus lpAdvanceWidth has both a horizontal and vertical component.

1.3.3.4.3.16.5 IG_PDE_text_get_bbox

Gets the bounding box of a character or a text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_bbox(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPAT_PDF_FIXEDRECT lpBBox
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose bounding box is found.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpBBox	LPAT_PDF_FIXEDRECT	Pointer to AT_PDF_FIXEDRECT to set to the bounding box of specified character or text run.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.6 IG_PDE_text_get_font

Gets the font for a text character or element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_font(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPHIG_PDE_FONT lphFont
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose font is found.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lphFont	LPHIG_PDE_FONT	HIG_PDE_FONT return value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.7 IG_PDE_text_get_gstate

Gets the graphics state of a character or a text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_gstate(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPAT_PDE_GRAPHICSTATE lpGstate
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose graphics state is obtained.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpGstate	LPAT_PDE_GRAPHICSTATE	Pointer to a AT_PDE_GRAPHICSTATE structure with graphics state of specified character or text run.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Non-NULL objects in the graphic state, such as the fill and stroke color spaces, have their reference counts incremented by this function. Be sure to release these non-NULL objects when disposing of lpGstate.

1.3.3.4.3.16.8 IG_PDE_text_get_item

Obtains a text item from a text element at a given index position.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_item(
        HIG_PDE_TEXT hText,
        UINT nIndex,
        LPHIG_PDE_TEXTITEM lphTextItem
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object from which the text item is obtained.
nIndex	UINT	Index of the text item in hText.
lphTextItem	LPHIG_PDE_TEXTITEM	The text item object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.9 IG_PDE_text_get_matrix

Returns the matrix of a character or a text element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_matrix(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPAT_PDF_FIXEDMATRIX lpMatrix
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose matrix is obtained.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX that holds the matrix of specified character or text run.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.10 IG_PDE_text_get_byte_count

Gets the number of bytes occupied by the character code or text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_byte_count(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPUINT lpnByteCount
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose text is examined.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpnByteCount	LPUINT	Number of bytes occupied by the text run or character.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.11 IG_PDE_text_get_char_count

Gets the number of characters in a text object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_char_count(
        HIG_PDE_TEXT hText,
        LPUINT lpnCharCount
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose text is examined.
lpnCharCount	LPUINT	Total number of characters in hText.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.12 IG_PDE_text_get_runs_count

Gets the number of text runs (show strings) in a text object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_runs_count(
        HIG_PDE_TEXT hText,
        LPUINT lpnRunsCount
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object whose number of text runs is found.
lpnRunsCount	LPUINT	Number of text runs in hText.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.13 IG_PDE_text_get_quad

Gets the quad bounding the specified text run or character.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_quad(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPAT_PDF_FIXEDQUAD lpQuad
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose text is examined.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run In addition, if the IG_PDE_TEXT_GET_BOUNDS flag is set, this function uses the font descriptor's FontBBox, which is the smallest rectangle that encloses all characters in the font. The advance portion is based on the x-coordinates of the left and right sides of FontBBox and the advance width.
nIndex	UINT	Index of the character or text run in the text object.
lpQuad	LPAT_PDF_FIXEDQUAD	Pointer to AT_PDF_FIXEDQUAD that bounds the specified character or text run.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

The advance portion of the quad is based on the left side bearing and advance width.

1.3.3.4.3.16.14 IG_PDE_text_get_run_for_char

Gets the index of the text run that contains the nth character in a text object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_run_for_char(
        HIG_PDE_TEXT hText,
        UINT nCharIndex,
        LPUINT lpnRunIndex
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose text is examined.
nCharIndex	UINT	Number of the character to find in hText.
lpnRunIndex	LPUINT	Index of the text run with the specified character index in hText.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.15 IG_PDE_text_get_state

Gets the text state of a character or a text element.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_state(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPAT_PDE_TEXTSTATE lpTstate
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose text is examined.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: • IG_PDE_TEXT_CHAR - for a text character • IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpTstate	LPAT_PDE_TEXTSTATE	Pointer to AT_PDE_TEXTSTATE structure to fill with the text state of the specified character or text run.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.16 IG_PDE_text_get_stroke_matrix

Gets the stroke matrix of a character or a text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_stroke_matrix(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPAT_PDF_FIXEDMATRIX lpMatrix
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose text is examined.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX that holds the stroke matrix of specified character or text run. This matrix is the transformation for line widths when stroking. The h and v values of the matrix are ignored.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.17 IG_PDE_text_get_text

Gets the text for a text run or character.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_get_text(
       HIG_PDE_TEXT hText,
       UINT nFlags,
       UINT nIndex,
       LPBYTE lpText,
       LPUINT lpnTextLen
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run whose text is found.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: • IG_PDE_TEXT_CHAR - for a text character • IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpText	LPBYTE	Text of specified character or text run. lpText must be large enough to hold the returned text.
lpnTextLen	LPUINT	Number of bytes in text run or character.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.18 IG_PDE_text_is_at_point

Tests whether a point is on specified text.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_is_at_point(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPAT_PDF_FIXEDPOINT lpPoint,
        LPAT_BOOL lpbResult
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpPoint	LPAT_PDF_FIXEDPOINT	The point, specified in user space coordinates.
lpbResult	LPAT_BOOL	TRUE if the point is on the text; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Checks if the point is in a bounding box for hText.

1.3.3.4.3.16.19 IG_PDE_text_is_at_rect

Tests whether any part of a rectangle is on the specified text.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_is_at_rect(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPAT_PDF_FIXEDRECT lpFixedRect,
        LPAT_BOOL lpbResult
);
```

Arguments:

Name	**Type**	**Description**
hText	HIG_PDE_TEXT	Text object containing a character or text run.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpFixedRect	LPAT_PDF_FIXEDRECT	The rectangle, specified in user space coordinates.
lpbResult	LPAT_BOOL	TRUE if the text is on the rectangle; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.20 IG_PDE_text_remove

Removes characters or text runs from a text object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_remove(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        UINT nCount
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: • IG_PDE_TEXT_CHAR - for a text character • IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
nCount	UINT	Number of characters or text runs to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.21 IG_PDE_text_remove_items

Removes contiguous text items from a text element starting at a given index position.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_remove_items(
        HIG_PDE_TEXT hText,
        UINT nIndex,
        UINT nCount
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object from which the text items are removed.
nIndex	UINT	Index of the first text item in pdeText to remove.
nCount	UINT	The number of text items to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.22 IG_PDE_text_replace_chars

Replaces characters in a text object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_replace_chars(
        HIG_PDE_TEXT hText,
        UINT nFlags,
        UINT nIndex,
        LPBYTE lpText,
        UINT nTextLen
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a character or text run.
nFlags	UINT	enumIGPDETextFlags value that specifies whether index refers to the character offset from the beginning of the text object or the index of the text run in the text object. Must be either: ● IG_PDE_TEXT_CHAR - for a text character ● IG_PDE_TEXT_RUN - for a text run
nIndex	UINT	Index of the character or text run in the text object.
lpText	LPBYTE	Replacement text.
nTextLen	UINT	Number of bytes to replace.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function does not change the number of characters in the text object; extra characters are ignored.

1.3.3.4.3.16.23 IG_PDE_text_run_get_char_offset

Gets the character offset of the first character of the specified text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_get_char_offset(
        HIG_PDE_TEXT hText,
        UINT nRunIndex,
        LPUINT lpnOffset
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a text run.
nRunIndex	UINT	Index of the text run.
lpnOffset	LPUINT	Character offset of the first character of the specified text run in hText.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.24 IG_PDE_text_run_get_char_count

Gets the number of characters in a text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_get_char_count(
        HIG_PDE_TEXT hText,
        UINT nRunIndex,
        LPUINT lpnCount
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a text run.
nRunIndex	UINT	Index of the text run.
lpnCount	LPUINT	Number of characters in the specified text run.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.25 IG_PDE_text_run_set_font

Sets the font of a text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_font(
        HIG_PDE_TEXT hText,
        UINT nRunIndex,
        HIG_PDE_FONT hFont
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a text run.
nRunIndex	UINT	Index of the text run.
hFont	HIG_PDE_FONT	Font set for the text run.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.26 IG_PDE_text_run_set_gstate

Sets the graphics state of a text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_gstate(
        HIG_PDE_TEXT hText,
        UINT nRunIndex,
        LPAT_PDE_GRAPHICSTATE lpGstate
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a text run.
nRunIndex	UINT	Index of the text run.
lpGstate	LPAT_PDE_GRAPHICSTATE	Pointer to AT_PDE_GRAPHICSTATE structure with graphics state to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.27 IG_PDE_text_run_set_matrix

Sets the matrix of a text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_matrix(
        HIG_PDE_TEXT hText,
        UINT nRunIndex,
        LPAT_PDF_FIXEDMATRIX lpMatrix
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a text run.
nRunIndex	UINT	Index of the text run.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX structure with matrix to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.28 IG_PDE_text_run_set_state

Sets the text state of a text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_state(
        HIG_PDE_TEXT hText,
        UINT nRunIndex,
        LPAT_PDE_TEXTSTATE lpState
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a text run.
nRunIndex	UINT	Index of the text run.
lpState	LPAT_PDE_TEXTSTATE	Pointer to AT_PDE_TEXTSTATE structure with state to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.29 IG_PDE_text_run_set_stroke_matrix

Sets the stroke matrix of a text run.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_run_set_stroke_matrix(
        HIG_PDE_TEXT hText,
        UINT nRunIndex,
        LPAT_PDF_FIXEDMATRIX lpMatrix
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a text run.
nRunIndex	UINT	Index of the text run.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX structure with store matrix to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.16.30 IG_PDE_text_split_run_at

Splits a text run into two text runs.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_text_split_run_at(
        HIG_PDE_TEXT hText,
        UINT nSplitLoc
);
```

Arguments:

Name	Type	Description
hText	HIG_PDE_TEXT	Text object containing a text run.
nSplitLoc	UINT	Split location, relative to the text object. The first text run is from character index 0 up to nSplitLoc. The second text run is from nSplitLoc + 1 to the end of the run.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17 HIG_PDE_TEXTITEM

Handle to the PDE text item object. TextItem - a PDE element representing a text object.

Members:

IG_PDE_textitem_create	Creates a text item element containing a character or text run, which can be added to a PDE Text object.
IG_PDE_textitem_copy_text	Copies the text from a text item element into a character buffer.
IG_PDE_textitem_get_font	Gets the font for a text item.
IG_PDE_textitem_set_font	Sets a font for a text item.
IG_PDE_textitem_get_gstate	Gets the graphics state for a text item.
IG_PDE_textitem_set_gstate	Sets the graphics state of a text item.
IG_PDE_textitem_get_text_length	Gets the text length for a text item.
IG_PDE_textitem_get_matrix	Gets the text matrix for a character in a text item.
IG_PDE_textitem_set_matrix	Sets the text matrix for a text item.
IG_PDE_textitem_get_state	Gets the text state of a text item.
IG_PDE_textitem_set_state	Sets the text state of a text item.
IG_PDE_textitem_remove_chars	Removes contiguous characters from a text item.
IG_PDE_textitem_replace_chars	Replaces characters in a text item.
IG_PDE_textitem_replace_text	Replaces all of the text in a text item.

1.3.3.4.3.17.1 IG_PDE_textitem_create

Creates a text item element containing a character or text run, which can be added to a PDE Text object.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_create(
        LPBYTE lpText,
        UINT nTextLen,
        HIG_PDE_FONT hFont,
        LPAT_PDE_GRAPHICSTATE lpGstate,
        LPAT_PDE_TEXTSTATE lpTstate,
        LPAT_PDF_FIXEDMATRIX lpTextMatrix
        LPHIG_PDE_TEXTITEM lphTextItem
);
```

Arguments:

Name	Type	Description
lpText	LPBYTE	Pointer to the characters to add. Passing NULL for text can invalidate the text object but will not raise an error. Callers must not pass NULL for this parameter.
nTextLen	UINT	Length of the text, in bytes.
hFont	HIG_PDE_FONT	Font for the element.
lpGstate	LPAT_PDE_GRAPHICSTATE	Pointer to AT_PDE_GRAPHICSTATE structure with the graphics state for the element.
lpTstate	LPAT_PDE_TEXTSTATE	Pointer to AT_PDE_TEXTSTATE structure with text state for the element.
lpTextMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX that holds the matrix for the element.
lphTextItem	LPHIG_PDE_TEXTITEM	A text element object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose the created object when finished with it.

1.3.3.4.3.17.2 IG_PDE_textitem_copy_text

Copies the text from a text item element into a character buffer.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_copy_text(
        HIG_PDE_TEXTITEM hTextItem,
        LPBYTE lpText,
        UINT nTextLen,
        LPUINT lpnTextItemLen
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXTITEM	Text item object.
lpText	LPBYTE	A pointer to a buffer in which to store the copy.
nTextLen	UINT	Length of the text buffer, in bytes.
lpnTextItemLen	LPUINT	The length in bytes of hTextItem.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.3 IG_PDE_textitem_get_font

Gets the font for a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_font(
        HIG_PDE_TEXT hTextItem,
        LPHIG_PDE_FONT lphFont
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXT	Text item whose font is obtained.
lphFont	LPHIG_PDE_FONT	Font for hTextItem.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.4 IG_PDE_textitem_set_font

Sets a font for a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_set_font(
        HIG_PDE_TEXT hTextItem,
        HIG_PDE_FONT hFont
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXT	Text object containing a text run.
hFont	HIG_PDE_FONT	Font set for a text item.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.5 IG_PDE_textitem_get_gstate

Gets the graphics state for a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_gstate(
        HIG_PDE_TEXT hTextItem,
        LPAT_PDE_GRAPHICSTATE lpGstate
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXT	Text item whose graphic state is obtained.
lpGstate	LPAT_PDE_GRAPHICSTATE	Pointer to AT_PDE_GRAPHICSTATE structure with graphics state of the text item.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Non-NULL objects in the graphic state, such as the fill and stroke color spaces, have their reference counts incremented by this function. Be sure to release these non-NULL objects when disposing of lpGstate.

1.3.3.4.3.17.6 IG_PDE_textitem_set_gstate

Sets the graphics state of a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_set_gstate(
        HIG_PDE_TEXTITEM hTextItem,
        LPAT_PDE_GRAPHICSTATE lpGstate
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXTITEM	Text item whose graphic state is set.
lpGstate	LPAT_PDE_GRAPHICSTATE	Pointer to AT_PDE_GRAPHICSTATE structure with graphics state to set.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.7 IG_PDE_textitem_get_text_length

Gets the text length for a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_text_length(
        HIG_PDE_TEXT hTextItem,
        LPLONG lpnLength
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXTITEM	Text item whose length is obtained.
lpnLength	LPLONG	The text length, in bytes.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.8 IG_PDE_textitem_get_matrix

Gets the text matrix for a character in a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_matrix(
        HIG_PDE_TEXTITEM hTextItem,
        UINT nCharOffset,
        LPAT_PDF_FIXEDMATRIX lpMatrix
);
```

Arguments:

Name	**Type**	**Description**
hTextItem	HIG_PDE_TEXTITEM	The text item.
nCharOffset	UINT	The offset of the character whose text matrix is obtained.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX with text matrix of the character.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.9 IG_PDE_textitem_set_matrix

Sets the text matrix for a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_set_matrix(
        HIG_PDE_TEXTITEM hTextItem,
        LPAT_PDF_FIXEDMATRIX lpMatrix
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXTITEM	The text item.
lpMatrix	LPAT_PDF_FIXEDMATRIX	Pointer to AT_PDF_FIXEDMATRIX with the new text matrix of the text item.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.10 IG_PDE_textitem_get_state

Gets the text state of a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_get_state(
        HIG_PDE_TEXTITEM hTextItem,
        LPAT_PDE_TEXTSTATE lpTstate
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXTITEM	The text item.
lpTstate	AT_PDE_TEXTSTATE	Pointer to AT_PDE_TEXTSTATE structure with text state of the text item..

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.11 IG_PDE_textitem_set_state

Sets the text state of a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_set_state(
        HIG_PDE_TEXTITEM hTextItem,
        LPAT_PDE_TEXTSTATE lpTstate
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXTITEM	The text item.
lpTstate	LPAT_PDE_TEXTSTATE	Pointer to AT_PDE_TEXTSTATE structure with new text state of the text item.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.12 IG_PDE_textitem_remove_chars

Removes contiguous characters from a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_remove_chars (
        HIG_PDE_TEXTITEM hTextItem,
        UINT nCharOffset,
        UINT nCount
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXTITEM	The text item.
nCharOffset	UINT	Offset of the first character to remove.
nCount	UINT	The number of characters to remove.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.17.13 IG_PDE_textitem_replace_chars

Replaces characters in a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_replace_chars (
        HIG_PDE_TEXTITEM hTextItem,
        UINT nCharIndex,
        LPBYTE lpNewChars,
        UINT nNewCharsLen
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXTITEM	The text item.
nCharIndex	UINT	Index position of the characters to replace.
lpNewChars	LPBYTE	Replacement text.
nNewCharsLen	UINT	Number of bytes to replace.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

This function does not change the number of characters in the text item; extra characters are ignored.

1.3.3.4.3.17.14 IG_PDE_textitem_replace_text

Replaces all of the text in a text item.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_textitem_replace_text (
        HIG_PDE_TEXTITEM hTextItem,
        LPBYTE lpNewText,
        UINT nNewTextLen
);
```

Arguments:

Name	Type	Description
hTextItem	HIG_PDE_TEXTITEM	The text item.
lpNewText	LPBYTE	Replacement text.
nNewTextLen	UINT	Number of bytes to replace.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.18 HIG_PDE_XGROUP

Handle to the PDE XGroup object. XGroup - a transparency (XGroup) resource.

Members:

IG_PDE_xgroup_create	Creates a new XGroup of the given type.
IG_PDE_xgroup_get_colorspace	Acquires the color space of the transparency group.
IG_PDE_xgroup_set_colorspace	Sets the color space for the XGroup.
IG_PDE_xgroup_get_isolated	Gets the isolated Boolean value of the transparency group.
IG_PDE_xgroup_set_isolated	Sets the XGroup to be isolated or not.
IG_PDE_xgroup_get_knockout	Gets the knockout Boolean value of the transparency group.
IG_PDE_xgroup_set_knockout	Sets the knockout value.

1.3.3.4.3.18.1 IG_PDE_xgroup_create

Creates a new XGroup of the given type.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_create(
        HIG_PDF_DOC hDoc,
        LONG nType,
        LPHIG_PDE_XGROUP lphXGroup
);
```

Arguments:

Name	Type	Description
hDoc	HIG_PDF_DOC	The document in which the object will be created.
nType	LONG	enumIGPDEXGroupCreateFlags value.
lphXGroup	LPHIG_PDE_XGROUP	The newly created XGroup object.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose the created object when finished with it.

1.3.3.4.3.18.2 IG_PDE_xgroup_get_colorspace

Acquires the color space of the transparency group.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_get_colorspace (
        HIG_PDE_XGROUP hXGroup,
        LPHIG_PDE_COLORSPACE lphColorSpace
);
```

Arguments:

Name	Type	Description
hXGroup	HIG_PDE_XGROUP	The transparency group object
lphColorSpace	LPHIG_PDE_COLORSPACE	The color space.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.18.3 IG_PDE_xgroup_set_colorspace

Sets the color space for the XGroup.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_set_colorspace (
        HIG_PDE_XGROUP hXGroup,
        HIG_PDE_COLORSPACE hColorSpace
);
```

Arguments:

Name	Type	Description
hXGroup	HIG_PDE_XGROUP	The transparency group object.
hColorSpace	HIG_PDE_COLORSPACE	The color space to associate with the XGroup.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.18.4 IG_PDE_xgroup_get_isolated

Gets the isolated Boolean value of the transparency group.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_get_isolated (
        HIG_PDE_XGROUP hXGroup,
        LPAT_PDF_BOOL lpbIsolated
);
```

Arguments:

Name	Type	Description
hXGroup	HIG_PDE_XGROUP	The transparency group object.
lpbIsolated	LPAT_PDF_BOOL	TRUE if the transparency group is isolated; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.18.5 IG_PDE_xgroup_set_isolated

Sets the XGroup to be isolated or not.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_set_isolated (
        HIG_PDE_XGROUP hXGroup,
        AT_PDF_BOOL bIsolated
);
```

Arguments:

Name	Type	Description
hXGroup	HIG_PDE_XGROUP	The transparency group object
bIsolated	AT_PDF_BOOL	TRUE to isolate the XGroup; FALSE otherwise.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Corresponds to the /I key within the XGroup's dictionary.

1.3.3.4.3.18.6 IG_PDE_xgroup_get_knockout

Gets the knockout Boolean value of the transparency group.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_get_knockout(
        HIG_PDE_XGROUP hXGroup,
        LPAT_PDF_BOOL lpbKnockout
);
```

Arguments:

Name	Type	Description
hXGroup	HIG_PDE_XGROUP	The transparency group object.
lpbKnockout	LPAT_PDF_BOOL	The knockout value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.18.7 IG_PDE_xgroup_set_knockout

Sets the knockout value.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xgroup_set_knockout(
        HIG_PDE_XGROUP hXGroup,
        AT_PDF_BOOL bKnockout
);
```

Arguments:

Name	Type	Description
hXGroup	HIG_PDE_XGROUP	The transparency group object.
bKnockout	AT_PDF_BOOL	The knockout value.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

1.3.3.4.3.19 HIG_PDE_XOBJECT

Handle to the PDE XObject object. XObject - an element representing an arbitrary XObject.

Members:

IG_PDE_xobject_create	Creates a new XObject from hObject.

1.3.3.4.3.19.1 IG_PDE_xobject_create

Creates a new XObject from hObject.

Declaration:

```
AT_ERRCOUNT ACCUAPI IG_PDE_xobject_create(
        HIG_PDF_BASOBJ hBasObj,
        LPHIG_PDE_XOBJECT lphXObject
);
```

Arguments:

Name	Type	Description
hBasObj	HIG_PDF_BASOBJ	Base object for XObject.
lphXObject	LPHIG_PDE_XOBJECT	XObject corresponding to the hBasObj.

Return Value:

Error count.

Supported Raster Image Formats:

This function does not process image pixels.

Remarks:

Call IG_PDE_element_release to dispose the created object when finished with it.

1.3.3.5 PDF Component Structures Reference

This section provides information about simple structure types that are used for creating, attributing, and manipulating general and editing objects.

The following table describes the structures supported by the ImageGear PDF component:

Subsidiary Type / DLL Structure	**Description**
AT_PDE_COLORDATA	Fill in one of the following members of this union, then pass it to PDE ColorSpace creation routine. Please see section 7.10 of the PDF Reference Manual for information on color spaces.
AT_PDE_COLORDATA_CALGRAY	CalGray color space.
AT_PDE_COLORDATA_CALRGB	CalRGB color space.
AT_PDE_COLORDATA_DEVICEN	DeviceN color space.
AT_PDE_COLORDATA_ICCBASED	ICC based color space.
AT_PDE_COLORDATA_INDEXED	Indexed color space.
AT_PDE_COLORDATA_LAB	L*a*b* color space.
AT_PDE_COLORDATA_SEPARATION	Separation color space
AT_PDE_COLORRANGE	Color range.
AT_PDE_COLORSPEC	Color specification.
AT_PDE_COLORVALUE	Color value.
AT_PDE_CONTENTATTRS	Attributes of a PDE Content object.
AT_PDE_DASH	Dash specification, as described in Table 4.8 in the PDF Reference. See Section 4.3.2 for more information on line dash patterns.
AT_PDE_FILTERARRAY	Filter information for streams. Array of FilterSpec elements. Usually consists of 2 filter elements: text encoding and image compression.
AT_PDE_FILTERSPEC	Filter element in a filter array.
AT_PDE_FONT_CREATEFROMSYSFONTPARAMS	Parameters for PDE font creation.
AT_PDE_FONT_INFO	PDE Font information.
AT_PDE_FONTATTRS	Attributes of a PDE Font and a PDF SysFont.
AT_PDE_GRAPHICSTATE	Attributes of a PDE Element or a PDE Text sub-element.
AT_PDE_IMAGEATTRS	Attributes of a PDE Image object.
AT_PDE_PSATTRS	Attributes of a PDE PS object.
AT_PDE_TEXTSTATE	Attributes of a PDE Text element.
AT_PDE_XYZCOLOR	XYZ color.
AT_PDF_BOOL	Boolean type with two values: TRUE (1) or FALSE (0).
AT_PDF_COLORVALUE	Data structure representing a color.

AT_PDF_FIXED	The Fixed type is a 32-bit quantity representing a rational number with the high (low on little-endian machines) 16 bits representing the number's mantissa and the low (high) 16 bits representing the fractional part.
AT_PDF_FIXEDMATRIX	Matrix containing fixed numbers.
AT_PDF_FIXEDPOINT	Point (in two-dimensional space) represented by two fixed numbers.
AT_PDF_FIXEDQUAD	Quadrilateral represented by four fixed points (one at each corner). A quadrilateral differs from a rectangle in that the latter must always have horizontal and vertical sides, and opposite sides must be parallel.
AT_PDF_FIXEDRECT	A rectangle represented by the coordinates of its four sides. A rectangle differs from a quadrilateral in that the former must always have horizontal and vertical sides, and opposite sides must be parallel.
AT_PDF_FLATTEN	Controls tile flattening.
AT_PDF_FONT_METRICS	Font metrics.
AT_PDF_FONT_STYLES	Font styles.
AT_PDF_PRINTOPTIONS	This structure is used to provide printing parameters for the IG_PDF_doc_print function.
AT_PDF_PRINTPARAMS	This structure indicates how a document should be printed.
AT_PDF_SECURITYDATA	Describes the data for the standard security handler.
AT_PDF_SYSFONT_PLATDATA	PDF SysFont platform specific data.
AT_PDF_TILE	Specifies printing flags.
AT_PDF_TILEEX	Specifies printing flags.

1.3.3.5.1 AT_PDE_COLORDATA

Fill in one of the following members of this union then pass this data to PDE color space creation routine.

Declaration:

```
typedef union tagAT_PDE_COLORDATA
{
        AT_PDE_COLORDATA_CALGRAY* calGray;
        AT_PDE_COLORDATA_CALRGB* calRGB;
        AT_PDE_COLORDATA_LAB* lab;
        AT_PDE_COLORDATA_ICCBASED* icc;
        AT_PDE_COLORDATA_INDEXED* indexed;
        HIG_PDE_COLORSPACE patternbase;
        AT_PDE_COLORDATA_SEPARATION* sep;
        AT_PDE_COLORDATA_DEVICEN* devn;
} AT_PDE_COLORDATA;
typedef AT_PDE_COLORDATA FAR* LPAT_PDE_COLORDATA;
```

Members:

Name	Type	Description
*calGray	AT_PDE_COLORDATA_CALGRAY*	Pointer to a structure describing a CalGray color space.
*calRGB	AT_PDE_COLORDATA_CALRGB*	Pointer to a structure describing a CalRGB color space.
*lab	AT_PDE_COLORDATA_LAB*	Pointer to a structure describing a L*a*b* color space.
*icc	AT_PDE_COLORDATA_ICCBASED*	Pointer to a structure describing an ICCBased color space.
*indexed	AT_PDE_COLORDATA_INDEXED*	Pointer to a structure describing an Indexed color space.
patternbase	HIG_PDE_COLORSPACE	A handle to a Pattern color space.
*sep	AT_PDE_COLORDATA_SEPARATION*	Pointer to a structure describing a Separation color space.
*devn	AT_PDE_COLORDATA_DEVICEN*	Pointer to a structure describing a DeviceN color space.

Remarks:

Please see section 7.10 of the PDF Reference Manual for information on color spaces.

1.3.3.5.2 AT_PDE_COLORDATA_CALGRAY

Describes a CalGray color space.

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_CALGRAY
{
        AT_PDE_XYZCOLOR whitePoint;
        AT_PDE_XYZCOLOR blackPoint;
        float gamma;
} AT_PDE_COLORDATA_CALGRAY;
typedef AT_PDE_COLORDATA_CALGRAY FAR* LPAT_PDE_COLORDATA_CALGRAY;
```

Members:

Name	**Type**	**Description**
whitePoint	AT_PDE_XYZCOLOR	White point
blackPoint	AT_PDE_XYZCOLOR	Black point
gamma	float	Gamma

Remarks:

Default calGray = {{0, 0, 0}, {0, 0, 0}, 1};

1.3.3.5.3 AT_PDE_COLORDATA_CALRGB

Describes a CalRGB color space.

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_CALRGB
{
        AT_PDE_XYZCOLOR whitePoint;
        AT_PDE_XYZCOLOR blackPoint;
        float redGamma;
        float greenGamma;
        float blueGamma;
        float matrix[9];
} AT_PDE_COLORDATA_CALRGB;
typedef AT_PDE_COLORDATA_CALRGB FAR* LPAT_PDE_COLORDATA_CALRGB;
```

Members:

Name	Type	Description
whitePoint	AT_PDE_XYZCOLOR	White point
blackPoint	AT_PDE_XYZCOLOR	Black point
redGamma	float	Red gamma
greenGamma	float	Green gamma
blueGamma	float	Blue gamma
matrix	float[9]	Matrix

Remarks:

Default calRGB = {{0, 0, 0}, {0, 0, 0}, 1, 1, 1, {1, 0, 0, 0, 1, 0, 0, 0, 1}};

1.3.3.5.4 AT_PDE_COLORDATA_DEVICEN

DeviceN color space.

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_DEVICEN
{
        AT_UINT size;
        HIG_PDF_ATOM* names;
        UINT nNames;
        HIG_PDE_COLORSPACE alt;
        HIG_PDF_BASOBJ tintTransform;
} AT_PDE_COLORDATA_DEVICEN;
typedef AT_PDE_COLORDATA_DEVICEN FAR* LPAT_PDE_COLORDATA_DEVICEN;
```

Members:

Name	Type	Description
size	AT_UINT	size = sizeof(AT_PDE_COLORDATA_DEVICEN).
*names	HIG_PDF_ATOM*	Names of colorants.
nNames	UINT	Number of colorants.
alt	HIG_PDE_COLORSPACE	Alternative color space.
tintTransform	HIG_PDF_BASOBJ	The tintTransform dictionary or function. See Section 4.5.5 in the PDF Reference for more information.

1.3.3.5.5 AT_PDE_COLORDATA_ICCBASED

ICC based color space.

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_ICCBASED
{
        AT_UINT size;
        HIG_PDF_STREAM iccstream;
        UINT nComps;
        HIG_PDE_COLORSPACE altCs;
} AT_PDE_COLORDATA_ICCBASED;
typedef AT_PDE_COLORDATA_ICCBASED FAR* LPAT_PDE_COLORDATA_ICCBASED;
```

Members:

Name	**Type**	**Description**
size	AT_UINT	size = sizeof(AT_PDE_COLORDATA_ICCBASED).
iccstream	HIG_PDF_STREAM	Stream containing ICC Profile.
nComps	UINT	Number of color components (1, 3, or 4).
altCs	HIG_PDE_COLORSPACE	Alternate ColorSpace (optional).

1.3.3.5.6 AT_PDE_COLORDATA_INDEXED

Indexed color space.

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_INDEXED
{
        AT_UINT size;
        HIG_PDE_COLORSPACE baseCs;
        WORD hival;
        LPBYTE lookup;
        UINT lookupLen;
} AT_PDE_COLORDATA_INDEXED;
typedef AT_PDE_COLORDATA_INDEXED FAR* LPAT_PDE_COLORDATA_INDEXED;
```

Members:

Name	Type	Description
size	AT_UINT	size = sizeof(AT_PDE_COLORDATA_INDEXED).
baseCs	HIG_PDE_COLORSPACE	Base colorspace.
hival	WORD	Highest color value.
lookup	LPBYTE	Indexed color lookup data.
lookupLen	UINT	Number of bytes in lookup data.

1.3.3.5.7 AT_PDE_COLORDATA_LAB

Describes a L*a*b* color space.

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_LAB
{
        AT_PDE_XYZCOLOR whitePoint;
        AT_PDE_XYZCOLOR blackPoint;
        AT_PDE_COLORRANGE rangeA, rangeB;
} AT_PDE_COLORDATA_LAB;
typedef AT_PDE_COLORDATA_LAB FAR* LPAT_PDE_COLORDATA_LAB;
```

Members:

Name	Type	Description
whitePoint	AT_PDE_XYZCOLOR	White point
blackPoint	AT_PDE_XYZCOLOR	Black point
rangeA	AT_PDE_COLORRANGE	Color ranges
rangeB	AT_PDE_COLORRANGE	Color ranges

Remarks:

Default lab = {{0, 0, 0}, {0, 0, 0}, {-100, 100}, {-100, 100}};

1.3.3.5.8 AT_PDE_COLORDATA_SEPARATION

Separation color space.

Declaration:

```
typedef struct tagAT_PDE_COLORDATA_SEPARATION
{
        AT_UINT size;
        HIG_PDF_ATOM name;
        HIG_PDE_COLORSPACE alt;
        HIG_PDF_BASOBJ tintTransform;
} AT_PDE_COLORDATA_SEPARATION;
typedef AT_PDE_COLORDATA_SEPARATION FAR* LPAT_PDE_COLORDATA_SEPARATION;
```

Members:

Name	Type	Description
size	AT_UINT	size = sizeof(AT_PDE_COLORDATA_SEPARATION).
name	HIG_PDF_ATOM	Name of separation or colorant.
alt	HIG_PDE_COLORSPACE	Alternative color space.
tintTransform	HIG_PDF_BASOBJ	The tintTransform dictionary or function. See Section 4.5.5 in the PDF Reference for more information.

1.3.3.5.9 AT_PDE_COLORRANGE

Contains color range.

Declaration:

```
typedef struct tagAT_PDE_COLORRANGE
{
        float min;
        float max;
} AT_PDE_COLORRANGE;
typedef AT_PDE_COLORRANGE FAR* LPAT_PDE_COLORRANGE;
```

Members:

Name	Type	Description
min	float	Minimum value
max	float	Maximum value

1.3.3.5.10 AT_PDE_COLORSPEC

Describes color specification.

Declaration:

```
typedef struct tagAT_PDE_COLORSPEC
{
        HIG_PDE_COLORSPACE space;
        AT_PDE_COLORVALUE value;
} AT_PDE_COLORSPEC;
typedef AT_PDE_COLORSPEC FAR* LPAT_PDE_COLORSPEC;
```

Members:

Name	Type	Description
space	HIG_PDE_COLORSPACE	The specified color space.
value	AT_PDE_COLORVALUE	The color value.

1.3.3.5.11 AT_PDE_COLORVALUE

Describes color value.

Declaration:

```
typedef struct tagAT_PDE_COLORVALUE
{
        AT_PDF_FIXED color[7];
        HIG_PDE_OBJECT colorObj2;
        HIG_PDE_OBJECT colorObj;
} AT_PDE_COLORVALUE;
typedef AT_PDE_COLORVALUE FAR* LPAT_PDE_COLORVALUE;
```

Members:

Name	Type	Description
color	AT_PDF_FIXED[7]	Color value components. For instance, a Gray color space has 1 component, an RGB color space has 3 components, a CMYK has 4 components, and so on.
colorObj2	HIG_PDE_OBJECT	For DeviceN color space.
colorObj	HIG_PDE_OBJECT	For color spaces whose color values do not have numeric values, such as the Pattern and Separation color spaces.

1.3.3.5.12 AT_PDE_CONTENTATTRS

Attributes of a PDE Content object.

Declaration:

```
typedef struct tagAT_PDE_CONTENTATTRS
{
        AT_DWORD flags;
        AT_PDF_FIXED cacheDevice[8];
        LONG formType;
        AT_PDF_FIXEDRECT bbox;
        AT_PDF_FIXEDMATRIX matrix;
        HIG_PDF_BASOBJ XUID;
} AT_PDE_CONTENTATTRS;
typedef AT_PDE_CONTENTATTRS FAR* LPAT_PDE_CONTENTATTRS;
```

Members:

Name	Type	Description
flags	AT_DWORD	enumIGPDEContentFlags value.
cacheDevice	AT_PDF_FIXED[8]	CharProc attributes If flags has IG_PDE_SET_CACHE_DEVICE set, the first 6 cache device values contain the operands for the d1 (setcachdevice) page operator. If flags has IG_PDE_SET_CHAR_WIDTH set, cacheDevice contains 2 charwidth values.
formType	LONG	Form attributes Only used if HIG_PDE_CONTENT contains a Form XObject. Corresponds to FormType key in the XObject Form attributes dictionary.
bbox	AT_PDF_FIXEDRECT	Only used if HIG_PDE_CONTENT contains a Form. Bounding box of the HIG_PDE_CONTENT object. Corresponds to BBox key in the XObject Form attributes dictionary.
matrix	AT_PDF_FIXEDMATRIX	Only used if HIG_PDE_CONTENT contains a Form. Transformation matrix for the HIG_PDE_CONTENT object. Corresponds to Matrix key in the XObject Form attributes dictionary.
XUID	HIG_PDF_BASOBJ	Only used if HIG_PDE_CONTENT contains a Form. The form's XUID, an ID that uniquely identifies the form. Corresponds to XUID key in the XObject Form attributes dictionary.

1.3.3.5.13 AT_PDE_DASH

Describes dash specification, as described in Table 4.8 in the PDF Reference (see Section 4.3.2 for more information on line dash patterns).

Declaration:

```
typedef struct tagAT_PDE_DASH
{
        AT_PDF_FIXED dashPhase;
        LONG dashLen;
        AT_PDF_FIXED dashes[11];
} AT_PDE_DASH;
typedef AT_PDE_DASH FAR* LPAT_PDE_DASH;
```

Members:

Name	Type	Description
dashPhase	AT_PDF_FIXED	Dash phase. Phase is a number that specifies a distance in user space into the dash pattern at which to begin marking the path.
dashLen	LONG	Number of entries in the dash array, an element of the Border array.
dashes	AT_PDF_FIXED[11]	Dash array, which specifies distances in user space for the length of dashes and gaps.

1.3.3.5.14 AT_PDE_FILTERARRAY

Array of filter specifications usually containing two or less filters: encoding and/or compression.

Declaration:

```
typedef struct tagAT_PDE_FILTERARRAY
{
        LONG numFilters;
        AT_PDE_FILTERSPEC spec[2];
} AT_PDE_FILTERARRAY;
typedef AT_PDE_FILTERARRAY FAR* LPAT_PDE_FILTERARRAY;
```

Members:

Name	Type	Description
numFilters	LONG	Number of filters in the array.
spec	AT_PDE_FILTERSPEC[2]	Variable length array of filter spec.

1.3.3.5.15 AT_PDE_FILTERSPEC

Filter element in a filter array.

Declaration:

```
typedef struct tagAT_PDE_FILTERSPEC
{
        HIG_PDF_BASOBJ decodeParms;
        HIG_PDF_BASOBJ encodeParms;
        HIG_PDF_ATOM name;
} AT_PDE_FILTERSPEC;
typedef AT_PDE_FILTERSPEC FAR* LPAT_PDE_FILTERSPEC;
```

Members:

Name	Type	Description
decodeParms	HIG_PDF_BASOBJ	Parameters used by the decoding filters specified with the Filter key. Corresponds to the DecodeParms key in the stream dictionary. Must be set to NULL if AT_PDE_FILTERSPEC is specified but no decode parameters are specified. This can be done by zeroing the unused decode params. Required decode params for DCTDecode are Columns, Rows, and Colors.
encodeParms	HIG_PDF_BASOBJ	Parameters used when encoding the stream. Required for DCTDecode filter; optional for other filters. Must be set to NULL if AT_PDE_FILTERSPEC is specified but no encode parameters are specified. This can be done by zeroing the unused encode params.
name	HIG_PDF_ATOM	Filter name. Supported filters are: ASCIIHexDecode, ASCII85Decode, LZWDecode, DCTDecode, CCITTFaxDecode, RunLengthDecode, and FlateDecode.

1.3.3.5.16 AT_PDE_FONT_CREATEFROMSYSFONTPARAMS

Data structure used with PDE Font creation.

Declaration:

```
typedef struct tagAT_PDE_FONT_CREATEFROMSYSFONTPARAMS
{
        AT_DWORD structSize;
        AT_DWORD flags;
        HIG_PDF_ATOM snapshotName;
        AT_PDF_FIXED* mmDesignVec;
        AT_INT ctCodePage;
        HIG_PDF_ATOM encoding;
        LPVOID cosDoc;
} AT_PDE_FONT_CREATEFROMSYSFONTPARAMS;
typedef AT_PDE_FONT_CREATEFROMSYSFONTPARAMS FAR* LPAT_PDE_FONT_CREATEFROMSYSFONTPARAMS;
```

Members:

Name	Type	Description
structSize	AT_DWORD	Size of the data structure. Must be set to sizeof(AT_PDE_FONT_CREATEFROMSYSFONTPARAMS).
flags	AT_DWORD	A bit mask of the enumIGPDEFontCreateFlags.
snapshotName	HIG_PDF_ATOM	The name of a multiple master snapshot. See PDF Reference for more information on snapshots.
*mmDesignVec	AT_PDF_FIXED*	Pointer to multiple master font design vector.
ctCodePage	AT_INT	Used to select a specific code page supported by the font. When a non-zero code page is supplied, embedding must be turned on and the IG_PDE_FONT_ENCODE_BY_GID flag set.
encoding	HIG_PDF_ATOM	Used to specify which encoding to use with a CID font. Pass IG_PDF_ATOM_NULL to use the platform default.
cosDoc	LPVOID	Unused. Set to 0.

1.3.3.5.17 AT_PDE_FONT_INFO

PDE font information

Declaration:

```
typedef struct tagAT_PDE_FONT_INFO
{
        HIG_PDF_ATOM name;
        HIG_PDF_ATOM type;
        HIG_PDF_ATOM charSet;
        HIG_PDF_ATOM encoding;
        SHORT wMode;
} AT_PDE_FONT_INFO;
typedef AT_PDE_FONT_INFO FAR* LPAT_PDE_FONT_INFO;
```

Members:

Name	**Type**	**Description**
name	HIG_PDF_ATOM	HIG_PDF_ATOM for font name, as in "Times-Roman."
type	HIG_PDF_ATOM	HIG_PDF_ATOM for font type, "Type 1," "TrueType," and so on.
charSet	HIG_PDF_ATOM	HIG_PDF_ATOM for "Roman" or IG_PDF_ATOM_NULL. If "Roman," the characters must be a subset of the Adobe Standard Roman Character Set.
encoding	HIG_PDF_ATOM	HIG_PDF_ATOM for font encoding, as in WinAnsiEncoding.
wMode	SHORT	Writing mode: 0 = horizontal; 1 = vertical.

1.3.3.5.18 AT_PDE_FONTATTRS

Attributes for HIG_PDE_FONT and HIG_PDF_SYSFONT.

Declaration:

```
typedef struct tagAT_PDE_FONTATTRS
{
        HIG_PDF_ATOM name;
        HIG_PDF_ATOM type;
        HIG_PDF_ATOM charSet;
        HIG_PDF_ATOM encoding;
        UINT flags;
        AT_PDF_FIXEDRECT fontBBox;
        SHORT missingWidth;
        SHORT stemV;
        SHORT stemH;
        SHORT capHeight;
        SHORT xHeight;
        SHORT ascent;
        SHORT descent;
        SHORT leading;
        SHORT maxWidth;
        SHORT avgWidth;
        SHORT italicAngle;
        HIG_PDF_ATOM cidFontType;
        SHORT wMode;
        HIG_PDF_ATOM psName;
        HIG_PDF_ATOM platformName;
        HIG_PDF_ATOM lang;
        HIG_PDF_ATOM registry;
        HIG_PDF_ATOM ordering;
        LONG supplement;
        LONG cantEmbed;
        HIG_PDF_ATOM deltaEncoding;
        UINT protection;
        LONG packageType;
} AT_PDE_FONTATTRS;
typedef AT_PDE_FONTATTRS FAR* LPAT_PDE_FONTATTRS;
```

Members:

Name	Type	Description
name	HIG_PDF_ATOM	A HIG_PDF_ATOM for font name, as in "Times-Roman." Corresponds to the BaseFont key in the font dictionary of a PDF file (see Section 5.6.3 in the PDF Reference).
type	HIG_PDF_ATOM	A HIG_PDF_ATOM for font type, corresponding to the Subtype key in a font dictionary. May be "Type1," "TrueType," "MMType1," or "Type0."
charSet	HIG_PDF_ATOM	A HIG_PDF_ATOM for "Roman" or IG_PDF_ATOM_NULL. If "Roman," the characters must be a subset of the Adobe Standard Roman Character Set.
encoding	HIG_PDF_ATOM	A HIG_PDF_ATOM for font encoding. May be MacRomanEncoding, WinAnsiEncoding, or IG_PDF_ATOM_NULL. In the case of IG_PDF_ATOM_NULL, call PDSysFontGetEncoding to get more information about the encoding.
flags	UINT	Desired font flags, one or more of Font Flags. Use IG_PDF_SCRIPT, etc. to get flags.
fontBBox	AT_PDF_FIXEDRECT	Font bounding box in 1000 EM units.
missingWidth	SHORT	Width of missing character (.notdef).

stemV	SHORT	Vertical stem width.
stemH	SHORT	Horizontal stem width.
capHeight	SHORT	Capital height.
xHeight	SHORT	X height.
ascent	SHORT	Max ascender height.
descent	SHORT	Max descender depth.
leading	SHORT	Additional leading between lines.
maxWidth	SHORT	Maximum character width.
avgWidth	SHORT	Average character width.
italicAngle	SHORT	Italic angle in degrees, if any.
cidFontType	HIG_PDF_ATOM	CIDFontType0 or CIDFontType2.
wMode	SHORT	Writing mode. Must be one of 0 for horizontal writing or 1 for vertical writing.
psName	HIG_PDF_ATOM	HIG_PDF_ATOM representing the PostScript name of a TrueType font.
platformName	HIG_PDF_ATOM	The platform name.
lang	HIG_PDF_ATOM	HIG_PDF_ATOM representing the ISO 639 language code. These are available from http://www.iso.ch.
registry	HIG_PDF_ATOM	HIG_PDF_ATOM representing the CIDFont's Registry information, as in "gAdobe-Japan".
ordering	HIG_PDF_ATOM	HIG_PDF_ATOM representing the CIDFont's Ordering information, for example, "g1".
supplement	LONG	The SystemSupplement field from the CIDFont.
cantEmbed	LONG	A non-zero value means the font can't be embedded.
deltaEncoding	HIG_PDF_ATOM	The name of the base encoding; that is, the BaseEncoding entry in an encoding dictionary (see section 5.5.5 of the PDF Reference). The Differences entry of the encoding dictionary describes differences (deltas) from the base encoding.
protection	UINT	protection Allows setting one of the following bits to disable font embedding: IG_PDE_FONT_NO_EMBEDDING = 1: font should not be embedded. IG_PDE_FONT_NO_EDITABLE_EMBEDDING = 2: font should not be embedded for editing purposes.
packageType	LONG	enumIGPDFSysFontPackageType value.

1.3.3.5.19 AT_PDE_GRAPHICSTATE

Attributes of a PDE element or a PDE text sub-element.

Declaration:

```
typedef struct tagAT_PDE_GRAPHICSTATE
{
        UINT wasSetFlags;
        AT_PDE_COLORSPEC fillColorSpec;
        AT_PDE_COLORSPEC strokeColorSpec;
        AT_PDE_DASH dash;
        AT_PDF_FIXED lineWidth;
        AT_PDF_FIXED miterLimit;
        AT_PDF_FIXED flatness;
        LONG lineCap;
        LONG lineJoin;
        HIG_PDF_ATOM renderIntent;
        HIG_PDE_OBJECT extGState;
        AT_PDF_FIXEDMATRIX softMaskMatrix;
} AT_PDE_GRAPHICSTATE;
typedef AT_PDE_GRAPHICSTATE FAR* LPAT_PDE_GRAPHICSTATE;
```

Members:

Name	**Type**	**Description**
wasSetFlags	UINT	enumIGPDEGraphicStateWasSetFlags indicating if an attribute has been set. Support for these flags is not complete. For compatibility, you should set them, but do not depend on reading their values back. The intended use is with XObject Forms to indicate whether the value is inherited or explicitly set.
fillColorSpec	AT_PDE_COLORSPEC	Fill color specification. The default value is DeviceGray, IG_PDF_FIXED_ZERO.
strokeColorSpec	AT_PDE_COLORSPEC	Stroke color specification. The default value is DeviceGray, IG_PDF_FIXED_ZERO.
dash	AT_PDE_DASH	Dash specification. The default value is [0, 0].
lineWidth	AT_PDF_FIXED	Line width, corresponding to the w (setlinewidth) operator. The default value is IG_PDF_FIXED_ONE.
miterLimit	AT_PDF_FIXED	Miter limit, corresponding to the M (setmiterlimit) operator. The default value is IG_PDF_FIXED_TEN.
flatness	AT_PDF_FIXED	Line flatness, corresponding to the i (setflat) operator. The default value is IG_PDF_FIXED_ZERO.
lineCap	LONG	Line cap style, corresponding to the J (setlinecap) operator. The default value is 0.
lineJoin	LONG	Line join style, corresponding to the j (setlinejoin) operator. The default value is 0.
renderIntent	HIG_PDF_ATOM	A color rendering intent, corresponding to the Intent key in the image dictionary. The default value is 0.
extGState	HIG_PDE_OBJECT	An extended graphics, corresponding to the gs operator. The default value is NULL.
softMaskMatrix	AT_PDF_FIXEDMATRIX	The CTM at the time soft mask was established. The default value is identity matrix.

1.3.3.5.20 AT_PDE_IMAGEATTRS

Attributes of a PDE Image object.

Declaration:

```
typedef struct tagAT_PDE_IMAGEATTRS
{
        UINT flags;
        LONG width;
        LONG height;
        LONG bitsPerComponent;
        AT_PDF_FIXED decode[8];
        HIG_PDF_ATOM intent;
} AT_PDE_IMAGEATTRS;
typedef AT_PDE_IMAGEATTRS FAR* LPAT_PDE_IMAGEATTRS;
```

Members:

Name	**Type**	**Description**
flags	UINT	enumIGPDEImageAttrFlags indicating image attributes.
width	LONG	Width of the image, corresponding to the Width key in the image dictionary.
height	LONG	Height of the image, corresponding to the Height key in the image dictionary.
bitsPerComponent	LONG	Number of bits used to represent each color component in the image, corresponding to the BitsPerComponent key in the image dictionary.
decode	AT_PDF_FIXED[8]	An array of numbers specifying the mapping from sample values in the image to values appropriate for the current color space. These values correspond to the Decode key in the image dictionary.
intent	HIG_PDF_ATOM	Color rendering intent, corresponding to the Intent key in the image dictionary.

1.3.3.5.21 AT_PDE_PSATTRS

Attributes of a PDE PostScript object.

Declaration:

```
typedef struct tagAT_PDE_PSATTRS
{
        UINT flags;
} AT_PDE_PSATTRS;
typedef AT_PDE_PSATTRS FAR* LPAT_PDE_PSATTRS;
```

Members:

Name	Type	Description
flags	UINT	IG_PDE_PS_INLINE

1.3.3.5.22 AT_PDE_TEXTSTATE

Attributes of a PDE text element.

Declaration:

```
typedef struct tagAT_PDE_TEXTSTATE
{
        UINT wasSetFlags;
        AT_PDF_FIXED charSpacing;
        AT_PDF_FIXED wordSpacing;
        LONG renderMode;
        AT_PDF_FIXED fontSize;
        AT_PDF_FIXED hScale;
        AT_PDF_FIXED textRise;
} AT_PDE_TEXTSTATE;
typedef AT_PDE_TEXTSTATE FAR* LPAT_PDE_TEXTSTATE;
```

Members:

Name	Type	Description
wasSetFlags	UINT	enumIGPDEGraphicStateWasSetFlags indicating if an attribute has been set. Support for these flags is not complete. For compatibility, you should set them, but do not depend on reading their values back. The intended use is with XObject Forms to indicate whether the value is inherited or explicitly set. PDFEdit ignores the wasSetFlags flag, so you must initialize the AT_PDE_TEXTSTATE fields.
charSpacing	AT_PDF_FIXED	Character spacing was set, corresponding to the Tc operator.
wordSpacing	AT_PDF_FIXED	Word spacing, corresponding to the Tw operator.
renderMode	LONG	Text rendering mode, corresponding to the Tr operator.
fontSize	AT_PDF_FIXED	Default is 1.
hScale	AT_PDF_FIXED	Default=100 (==100%)
textRise	AT_PDF_FIXED	Specifies the distance, in text space units that are not scaled, to move the baseline up or down from its default location. See Section 5.2.6 in the PDF Reference.

1.3.3.5.23 AT_PDE_XYZCOLOR

XYZ color data.

Declaration:

```
typedef struct tagAT_PDE_XYZCOLOR
{
        float x;
        float y;
        float z;
} AT_PDE_XYZCOLOR;
typedef AT_PDE_XYZCOLOR FAR* LPAT_PDE_XYZCOLOR;
```

Members:

Name	Type	Description
x	float	The X component of a tristimulus value in CIE 1931 XYZ color space.
y	float	The Y component of a tristimulus value in CIE 1931 XYZ color space.
z	float	The Z component of a tristimulus value in CIE 1931 XYZ color space.

Remarks:

Please see section 7.10 of the PDF Reference Manual for information on color spaces.

1.3.3.5.24 AT_PDF_BOOL

Boolean type with two values: TRUE (1) or FALSE (0).

Declaration:

```
typedef WORD AT_PDF_BOOL;
typedef AT_PDF_BOOL FAR *LPAT_PDF_BOOL;
```

1.3.3.5.25 AT_PDF_COLORVALUE

Data structure representing a color.

Declaration:

```
typedef struct tagAT_PDF_COLORVALUE
{
       BYTE space;
       AT_PDF_FIXED value[4];

} AT_PDF_COLORVALUE;
typedef AT_PDF_COLORVALUE FAR* LPAT_PDF_COLORVALUE;
```

Members:

Name	**Type**	**Description**
space	BYTE	The color space type. Can be one of the following: ● IG_PDF_DEVICE_GRAY. Grayscale color specification. Requires 1 value entry to specify the color. ● IG_PDF_DEVICE_RGB. Red-Green-Blue color specification. Requires 3 value entries to specify the color. ● IG_PDF_DEVICE_CMYK. Cyan-Magenta-Yellow-Black color specification. Requires 4 value entries to specify the color.
value	AT_PDF_FIXED[4]	The color value. The number of elements needed in the value field depends on the color space type (specified in the space field): ● IG_PDF_DEVICE_GRAY - 1 value ● IG_PDF_DEVICE_RGB - 3 values ● IG_PDF_DEVICE_CMYK - 4 values

1.3.3.5.26 AT_PDF_FIXED

The Fixed type is a 32-bit quantity representing a rational number with the high (low on little-endian machines) 16 bits representing the number's mantissa and the low (high) 16 bits representing the fractional part.

Declaration:

```
typedef LONG AT_PDF_FIXED;
typedef AT_PDF_FIXED FAR *LPAT_PDF_FIXED;
```

Remarks:

The definition is platform-dependent. Addition, subtraction, and negation with AT_PDF_FIXED types can be done with + and -, unless you care about overflow. Overflow in Fixed-value operations is indicated by the values IG_PDF_FIXED_POSITIVE_INFINITY and IG_PDF_FIXED_NEGATIVE_INFINITY.

1.3.3.5.27 AT_PDF_FIXEDMATRIX

Matrix containing fixed numbers.

Declaration:

```
typedef struct tagAT_PDF_FIXEDMATRIX
{
        AT_PDF_FIXED a;
        AT_PDF_FIXED b;
        AT_PDF_FIXED c;
        AT_PDF_FIXED d;
        AT_PDF_FIXED h;
        AT_PDF_FIXED v;
} AT_PDF_FIXEDMATRIX;
typedef AT_PDF_FIXEDMATRIX FAR* LPAT_PDF_FIXEDMATRIX;
```

Members:

Name	Type	Description
a	AT_PDF_FIXED	A value
b	AT_PDF_FIXED	B value
c	AT_PDF_FIXED	C value
d	AT_PDF_FIXED	D value
h	AT_PDF_FIXED	H value
v	AT_PDF_FIXED	V value

1.3.3.5.28 AT_PDF_FIXEDPOINT

Point (in two-dimensional space) represented by two fixed numbers.

Declaration:

```
typedef struct tagAT_PDF_FIXEDPOINT
{
        AT_PDF_FIXED h;
        AT_PDF_FIXED v;
} AT_PDF_FIXEDPOINT;
typedef AT_PDF_FIXEDPOINT FAR* LPAT_PDF_FIXEDPOINT;
```

Members:

Name	Type	Description
h	AT_PDF_FIXED	Horizontal value.
v	AT_PDF_FIXED	Vertical value.

1.3.3.5.29 AT_PDF_FIXEDQUAD

Quadrilateral represented by four fixed points (one at each corner); a quadrilateral differs from a rectangle in that the latter must always have horizontal and vertical sides, and opposite sides must be parallel.

Declaration:

```
typedef struct tagAT_PDF_FIXEDQUAD
{
        AT_PDF_FIXEDPOINT tl, tr, bl, br;
} AT_PDF_FIXEDQUAD;
typedef AT_PDF_FIXEDQUAD FAR* LPAT_PDF_FIXEDQUAD;
```

Members:

Name	**Type**	**Description**
tl	AT_PDF_FIXEDPOINT	Top left value
tr	AT_PDF_FIXEDPOINT	Top right value
bl	AT_PDF_FIXEDPOINT	Bottom left value
br	AT_PDF_FIXEDPOINT	Bottom right value

1.3.3.5.30 AT_PDF_FIXEDRECT

A rectangle represented by the coordinates of its four sides; a rectangle differs from a quadrilateral in that the former must always have horizontal and vertical sides, and opposite sides must be parallel.

Declaration:

```
typedef struct tagAT_PDF_FIXEDRECT
{
        AT_PDF_FIXED left;
        AT_PDF_FIXED top;
        AT_PDF_FIXED right;
        AT_PDF_FIXED bottom;
} AT_PDF_FIXEDRECT;
typedef AT_PDF_FIXEDRECT FAR* LPAT_PDF_FIXEDRECT;
```

Members:

Name	Type	Description
left	AT_PDF_FIXED	Left value
right	AT_PDF_FIXED	Right value
top	AT_PDF_FIXED	Top value
bottom	AT_PDF_FIXED	Bottom value

1.3.3.5.31 AT_PDF_FLATTEN

Controls tile flattening.

Declaration:

```
typedef struct tagAT_PDF_FLATTEN
{
     AT_UINT size;
     AT_INT32 tilingMode;
     AT_PDF_BOOL useTextOutlines;
     AT_PDF_BOOL allowShadingOutput;
     AT_PDF_BOOL allowLevel3ShadingOutput;
     AT_PDF_BOOL strokeToFill;
     AT_PDF_BOOL clipComplexRegions;
     AT_FLOAT internalDPI;
     AT_FLOAT externalDPI;
     AT_FLOAT pathDPI;
     AT_DWORD tileSizePts;
     AT_DWORD maxFltnrImageSize;
     AT_DWORD adaptiveThreshold;
     AT_PDF_BOOL preserveOverprint;

} AT_PDF_FLATTEN;
typedef AT_PDF_FLATTEN FAR* LPAT_PDF_FLATTEN;
```

Members:

Name	**Type**	**Description**
size	AT_UINT	Must be set to the size of this struct.
tilingMode	AT_INT32	Specifies the tiling mode. One of the following values: • 0 = no tiling • 1 = constant tiling • 2 = adaptive tiling
useTextOutlines	AT_PDF_BOOL	Outputs text outlines instead of native text when set to TRUE.
allowShadingOutput	AT_PDF_BOOL	Allows shading output when set to TRUE.
allowLevel3ShadingOutput	AT_PDF_BOOL	Allows Level 3 shading when set to TRUE.
strokeToFill	AT_PDF_BOOL	Converts stroke to outline when set to TRUE.
clipComplexRegions	AT_PDF_BOOL	Displays the Clip Complex checkbox when set to TRUE.
internalDPI	AT_FLOAT	Specifies the resolution for flattening the interior of atomic regions.
externalDPI	AT_FLOAT	Specifies the resolution for flattening the edges of atomic regions.
pathDPI	AT_FLOAT	Specifies the flattener path resolution; the default is 800.
tileSizePts	AT_DWORD	Specifies the target tile size, in points.
maxFltnrImageSize	AT_DWORD	Specifies the maximum image size when flattening; the default is 0.
adaptiveThreshold	AT_DWORD	Specifies the adaptive flattening threshold.
preserveOverprint	AT_PDF_BOOL	Attempts to preserve overprint when set to TRUE.

1.3.3.5.32 AT_PDF_FONT_METRICS

Font metrics.

Declaration:

```
typedef struct tagAT_PDF_FONT_METRICS
{
        UINT flags;
        AT_PDF_FIXEDRECT fontBBox;
        AT_INT16 missingWidth;
        AT_INT16 stemV;
        AT_INT16 stemH;
        AT_INT16 capHeight;
        AT_INT16 xHeight;
        AT_INT16 ascent;
        AT_INT16 descent;
        AT_INT16 leading;
        AT_INT16 maxWidth;
        AT_INT16 avgWidth;
        AT_INT16 italicAngle;
        AT_PDF_FONT_STYLES style;
        AT_INT16 baseLineAdj;
} AT_PDF_FONT_METRICS;
```

Members:

Name	Type	Description
flags	AT_PDF_FIXEDRECT	Must be an OR of the Font Flags values. All unused flags must be off.
fontBBox	AT_INT16	Font bounding box in 1000 EM units. (An EM is a typographic unit of measurement equal to the size of a font. In a 12-point font, an EM is 12 points.)
missingWidth	AT_INT16	Width of missing character.
stemV	AT_INT16	Vertical stem width.
stemH	AT_INT16	Horizontal stem width.
capHeight	AT_INT16	Capital height.
xHeight	AT_INT16	X height.
ascent	AT_INT16	Max ascender height.
descent	AT_INT16	Max descender depth.
Leading	AT_INT16	Additional leading between lines.
maxWidth	AT_INT16	Maximum character width.
avgWidth	AT_INT16	Average character width.
italicAngle	AT_INT16	Italic angle in degrees, if any.
style	AT_PDF_FONT_STYLES	Panose and sFamily class values.
baseLineAdj	AT_INT16	Baseline adjustment, which is a vertical adjustment for font baseline difference and writing mode 1 (vertical). This should only be used for CIDFontType 2 fonts with font substitution.

1.3.3.5.33 AT_PDF_FONT_STYLES

Font styles.

Declaration:

```
typedef struct tagAT_PDF_FONT_STYLES
{
        AT_BYTE sFamilyClassID;
        AT_BYTE sFamilySubclassID;
        AT_BYTE bFamilyType;
        AT_BYTE bSerifStyle;
        AT_BYTE bWeight;
        AT_BYTE bProportion;
} AT_PDF_FONT_STYLES;
```

Members:

Name	Type	Description
sFamilyClassID	AT_BYTE	Number that identifies the font family and determines the meaning of the remaining Panose digits. Possible families are Latin, Kanji, Hebrew, and so forth.
sFamilySubclassID	AT_BYTE	Number to identify the kind of family: text, decorative, handwritten, symbols, and so on.
bFamilyType	AT_BYTE	Number to identify the family type: text, decorative, handwritten, symbols, and so on.
bSerifStyle	AT_BYTE	Number that specifies the font's serif style, such as cove, obtuse cove, square, bone, and so forth.
bWeight	AT_BYTE	Number that specifies the font's weight, such as very light, heavy, black, and so on.
bProportion	AT_BYTE	Number that specifies the font's proportions, such as modern, expanded, condensed, mono-spaced, and so on.

1.3.3.5.34 AT_PDF_PRINTOPTIONS

This structure provides printing parameters for the IG_PDF_doc_print function.

Declaration:

```
typedef struct tagAT_PDF_PRINTOPTIONS
{
    AT_DWORD size;
    LPAT_PDF_PRINTPARAMS printParams;
    AT_PDF_BOOL emitToFile;
    HIG_PDF_STREAM printStm;
    AT_WORD paperWidth;
    AT_WORD paperHeight;
    AT_DWORD dontEmitListLen;
    char** dontEmitList;
    AT_PDF_BOOL emitToPrinter;
    char* command;
    LPVOID cancelProc;
    LPVOID clientData;
    int startResult;
    LPVOID userCallbacks;
    LONG startPage;
    LONG endPage;
    LONG psLevel;
    int nCopies;
    AT_UINT PPDFeatures;
    AT_UINT ppdFileName;
} AT_PDF_PRINTOPTIONS;
typedef AT_PDF_PRINTOPTIONS FAR* LPAT_PDF_PRINTOPTIONS;
```

Members:

Name	**Type**	**Description**
size	AT_DWORD	Size
printParams	LPAT_PDF_PRINTPARAMS	AT_PDF_PRINTPARAMS structure. Applies to PostScript file.
emitToFile	AT_PDF_BOOL	Create a PostScript file; must be FALSE for now.
printStm	HIG_PDF_STREAM	Writeable HIG_PDF_STREAM that points to file stm or proc stm.
paperWidth	AT_WORD	Width of paper in points.
paperHeight	AT_WORD	Height of paper in points.
dontEmitListLen	AT_DWORD	Number of fonts that should not be downloaded.
dontEmitList	char**	List of fonts (T1, TT, CID) that should not be downloaded.
emitToPrinter	AT_PDF_BOOL	Output PDF file to a PS or non-PS printer; must be TRUE for now.
command	char*	Optional command line arguments, used only if emitToPrinter is true. Example: "lp" or "lpr"
cancelProc	LPVOID	CancelProc and clientData are optional for emitToFile or emitToPrinter. LPFNIG_PDF_PRINTCANCELPROC callback function.
clientData	LPVOID	Optional data passed to cancelProc. Applies to both PostScript printer and file.

startResult	int	Spooler ID from StartDoc().
userCallbacks	LPVOID	Unused. Set to 0.
startPage	LONG	Page to start printing with, 0-based.
endPage	LONG	Page to end printing on.
psLevel	LONG	PostScript level.
nCopies	int	The number of copies to print.
PPDFeatures	AT_UINT	Unused. Set to 0.
ppdFileName	AT_UINT	Unused. Set to 0.

1.3.3.5.35 AT_PDF_PRINTPARAMS

This structure indicates how a document should be printed.

Declaration:

```
typedef struct tagAT_PDF_PRINTPARAMS
{
        AT_UINT size;
        AT_PDF_PAGE_RANGE* ranges;
        AT_INT32 numRanges;
        AT_PDF_BOOL shrinkToFit;
        AT_PDF_BOOL expandToFit;
        AT_PDF_BOOL rotateAndCenter;
        CHAR printWhat;
        CHAR printWhatAnnot;
        AT_PDF_BOOL emitPS;
        AT_INT32 psLevel;
        CHAR outputType;
        CHAR incBaseFonts;
        CHAR incEmbeddedFonts;
        CHAR incType1Fonts;
        CHAR incType3Fonts;
        CHAR incTrueTypeFonts;
        CHAR incCIDFonts;
        CHAR incProcsets;
        CHAR incOtherResources;
        AT_INT32 fontPerDocVM;
        AT_PDF_BOOL emitShowpage;
        AT_PDF_BOOL emitTTFontsFirst;
        AT_PDF_BOOL setPageSize;
        AT_PDF_BOOL emitDSC;
        AT_PDF_BOOL setupProcsets;
        AT_PDF_BOOL emitColorSeps;
        AT_PDF_BOOL binaryOK;
        AT_PDF_BOOL useSubFileDecode;
        AT_PDF_BOOL emitRawData;
        AT_PDF_BOOL TTasT42;
        AT_FLOAT scale;
        AT_PDF_BOOL emitExternalStreamRef;
        AT_PDF_BOOL emitHalftones;
        AT_PDF_BOOL emitPSXObjects;
        AT_PDF_BOOL centerCropBox;
        AT_PDF_BOOL emitSeparableImagesOnly;
        AT_PDF_BOOL emitDeviceExtGState;
        AT_PDF_FIXEDRECT boundingBox;
        AT_PDF_BOOL useFontAliasNames;
        AT_PDF_BOOL emitPageRotation;
        AT_PDF_BOOL reverse;
        AT_PDF_FIXEDRECT* tCropBox;
        AT_PDF_BOOL emitPageClip;
        AT_PDF_BOOL emitTransfer;
        AT_PDF_BOOL emitBG;
        AT_PDF_BOOL emitUCR;
        CHAR farEastFontOpt;
        AT_PDF_BOOL suppressCJKSubstitution;
        AT_PDF_BOOL suppressCSA;
        AT_PDF_BOOL hostBased;
        HIG_PDF_ATOM hostBasedOutputCS;
        CHAR duplex;
        CHAR doTiling;
        LPAT_PDF_TILEEX tileInfo;
        AT_PDF_BOOL rotate;
        AT_PDF_BOOL hostBasedCM;
        char destProfile[256];
        HIG_PDF_ATOM destCSAtom;
        AT_PDF_BOOL saveVM;
        AT_PDF_BOOL doOPP;
        AT_INT32 suppressOPPWhenNoSpots;
        AT_PDF_BOOL optimizeForSpeed;
        AT_PDF_BOOL brokenCRDs;
        AT_PDF_BOOL useMaxVM;
        AT_INT32 lastWidth;
        AT_INT32 lastHeight;
        AT_DWORD bitmapResolution;
        AT_DWORD gradientResolution;
        AT_DWORD transparencyQuality;
        AT_DWORD ocContext;
        AT_PDF_BOOL applyOCGPrintOverrides;
        AT_PDF_BOOL useFullResolutionJP2KData;
        AT_PDF_BOOL emitInRipSeps;
        AT_DWORD whichMarks;
        AT_PDF_BOOL westernMarksStyle;
        AT_PDF_BOOL doProofing;
        char proofProfile[256];
        AT_PDF_BOOL inkBlack;
        AT_PDF_BOOL paperWhite;
        AT_PDF_BOOL useExecForm;
        LPAT_PDF_FLATTEN flattenInfo;
        AT_PDF_BOOL negative;
        CHAR mirrorprint;
        AT_DWORD numCollatedCopies;
        AT_PDF_BOOL emitFlatness;
        AT_INT32 trapType;
        AT_PDF_BOOL TTasCIDT2;
        AT_DWORD markStyle;
        AT_FLOAT lineWidth;
        AT_PDF_BOOL macQDPrinter;
        AT_UINT customMarksFileName;
        AT_VOID* pAGMPI;
        AT_PDF_BOOL disableFlattening;
        AT_PDF_BOOL doNotDownloadFauxFonts;
        AT_PDF_BOOL suppressSnapToDevice;
        AT_INT32 suppressElement;
        AT_DWORD maxFlatSeconds;
        AT_DWORD testTilingMode;
} AT_PDF_PRINTPARAMS;
typedef AT_PDF_PRINTPARAMS FAR* LPAT_PDF_PRINTPARAMS;
```

Members:

Name	**Type**	**Description**
size	AT_UINT	Size of the data structure. Must be set to sizeof(AT_PDF_PRINTPARAMS).
ranges	AT_PDF_PAGE_RANGE*	Ranges of pages to print. Use NULL to print the entire document.
numRanges	AT_INT32	Number of ranges of pages to print in ranges. The default value is 0.
shrinkToFit	AT_PDF_BOOL	TRUE if the page is scaled to fit the printer page size; FALSE otherwise. This field overrides scale. The default value is FALSE.
expandToFit	AT_PDF_BOOL	TRUE if small pages are to be scaled up to fit the printer page size; FALSE otherwise. Overrides scale. The default value is FALSE.

rotateAndCenter	AT_PDF_BOOL	TRUE if page is to be rotated to fit printer's orientation, and centered in printer's page size; FALSE otherwise.The default value is FALSE. Rotation and centering (TRUE) only occur, however, if the page contents are too wide to fit on a narrow page (or vice versa) and the page contents are less than an inch smaller than the target page in one direction.
printWhat	CHAR	enumIGPDFPrintWhat flag. Default is IG_PDF_PRINT_DOCUMENT.
printWhatAnnot	CHAR	Combination of enumIGPDFPrintWhatAnnot flags which extend printWhat to enable Pro product behavior.
emitPS	AT_PDF_BOOL	If TRUE, emit a PostScript file. The default value is TRUE.
psLevel	AT_INT32	PostScript level: 1, 2 or 3. The default value is 2.
outputType	CHAR	Print PostScript or EPS with or without a preview.
incBaseFonts	CHAR	Embed the base fonts. The default value is IG_PDF_INCLUDE_NEVER.
incEmbeddedFonts	CHAR	Embed fonts that are embedded in the PDF file. This overrides the incType1Fonts, incTrueTypeFonts, and incCIDFonts fields. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
incType1Fonts	CHAR	Embed Type 1 fonts. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
incType3Fonts	CHAR	Embed Type 3 fonts. The default value is IG_PDF_INCLUDE_ON_EVERY_PAGE. This parameter must always be set to IG_PDF_INCLUDE_ON_EVERY_PAGE. PDF files exist with Type 3 fonts that contain different encodings on different pages.
incTrueTypeFonts	CHAR	Embed TrueType fonts. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
incCIDFonts	CHAR	Embed CID fonts. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
incProcsets	CHAR	Include Procsets in the file. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
incOtherResources	CHAR	Include all other types of resources in the file. The default value is IG_PDF_INCLUDE_ONCE_PER_DOC.
fontPerDocVM	AT_INT32	Amount of VM available for font downloading at the document level. Ignored if <=0. This must be set to 0 for the toolkit; it is only used by the viewer.
emitShowpage	AT_PDF_BOOL	Emit save and restore showpage in PostScript files. The default value is TRUE.
emitTTFontsFirst	AT_PDF_BOOL	Emit TrueType fonts before any other fonts. The default value is FALSE.
setPageSize	AT_PDF_BOOL	(PostScript level 2 only) Set the page size on each page. Use the media box for outputting to PostScript files, use the crop box for EPS files. Default is FALSE.
emitDSC	AT_PDF_BOOL	Write DSC (Document Structuring Conventions) comments. The default value is TRUE.
setupProcsets	AT_PDF_BOOL	If procsets are included, also include init/term code. The default value is TRUE. This must be set to TRUE.

emitColorSeps	AT_PDF_BOOL	Emit images for Level-1 separations. The default value is FALSE.
binaryOK	AT_PDF_BOOL	TRUE if binary data is permitted in the PostScript file; FALSE otherwise. The default value is TRUE.
useSubFileDecode	AT_PDF_BOOL	Add SubFileDecode filter to work around stream problems. The default value is FALSE.
emitRawData	AT_PDF_BOOL	TRUE if add no unnecessary filters when emitting image data; FALSE otherwise. The default value is TRUE.
TTasT42	AT_PDF_BOOL	If including TrueType fonts, convert to Type 42 fonts instead of Type 1 fonts. The default value is FALSE.
scale	AT_FLOAT	Document-wide scale factor. 100.0 = full size. The default value is 100.
emitExternalStreamRef	AT_PDF_BOOL	If an Image resource uses an external stream, emit code that points to the external file. The default value is FALSE. This must be set to FALSE.
emitHalftones	AT_PDF_BOOL	Preserve any halftone screening in the PDF file. The default value is FALSE.
emitPSXObjects	AT_PDF_BOOL	Emit PostScript XObjects into the PostScript stream. The default value is FALSE.
centerCropBox	AT_PDF_BOOL	TRUE if CropBox output is centered on the page when the CropBox < MediaBox; FALSE otherwise. The default value is TRUE.
emitSeparableImagesOnly	AT_PDF_BOOL	If emitting EPS, include only CMYK and gray images.
emitDeviceExtGState	AT_PDF_BOOL	When emitting the extended graphics state, include the device-dependent parameters (overprint, black generation, undercolor removal, transfer, halftone, halftone phase, smoothness, flatness, rendering intent) in addition to the device-independent parameters (font, line width, line cap, line join, miter limit, dash pattern). If this flag is FALSE, only the device-independent parameters will be emitted. This flag overrides emitHalftones; if this is FALSE, then halftones are not emitted. The default value is TRUE.
boundingBox	AT_PDF_FIXEDRECT	If all zeroes, is ignored. Otherwise, is used for %BoundingBox DSC comment and in centerCropBox calculations and for setpagedevice. The default value is [0 0 0 0].
useFontAliasNames	AT_PDF_BOOL	Used when printing with system fonts. The default value is FALSE.
emitPageRotation	AT_PDF_BOOL	Emit a concat at the beginning of each page so that the page is properly rotated. Used when emitting EPS. The default value is FALSE.
reverse	AT_PDF_BOOL	If set to TRUE, reverse the order of page output.
tCropBox	AT_PDF_FIXEDRECT*	Temporary crop box to represent selected region.
emitPageClip	AT_PDF_BOOL	Set to TRUE to emit page clip.
emitTransfer	AT_PDF_BOOL	Set to TRUE to emit transfer.
emitBG	AT_PDF_BOOL	Set to TRUE to emit black generation.
emitUCR	AT_PDF_BOOL	Set to TRUE to emit undercolor removal.
farEastFontOpt	CHAR	Far East font option. Currently not used. Set to 0.
suppressCJKSubstitution	AT_PDF_BOOL	If TRUE, do not do CJK substitution on the printer.
suppressCSA	AT_PDF_BOOL	If TRUE, don't emit CSAs for 4 component (CMYK) colors.
hostBased	AT_PDF_BOOL	For separator, do host-based color management.
hostBasedOutputCS	HIG_PDF_ATOM	The output color space when hostBased color management

		is TRUE.
duplex	CHAR	Currently not used. Set to 0.
doTiling	CHAR	Whether to tile none, all, or only large pages
tileInfo	LPAT_PDF_TILEEX	If non-NULL, tiling is desired with these parameters.
rotate	AT_PDF_BOOL	Enable the auto-rotating behavior from past versions of Acrobat.
hostBasedCM	AT_PDF_BOOL	Host base color management. Default: FALSE, do CSA generation for profiles instead of converting all colors on the host.
destProfile	char[256]	If hostBaseCM color management is TRUE, use this profile.
destCSAtom	HIG_PDF_ATOM	An atom representing the device color space (DeviceGray, DeviceRGB, etc.).
saveVM	AT_PDF_BOOL	TRUE means try to save VM when printing to PostScript.
doOPP	AT_PDF_BOOL	If TRUE, do the overprint preview operation.
suppressOPPWhenNoSpots	AT_INT32	When TRUE, suppress OPP for pages that do not contain spot colors
optimizeForSpeed	AT_PDF_BOOL	If TRUE, do it fast; FALSE means PostScript code must be page independent. If set to TRUE, font downloads are forced from IG_PDF_INCLUDE_ON_EVERY_PAGE to IG_PDF_INCLUDE_ONCE_PER_DOC.
brokenCRDs	AT_PDF_BOOL	If TRUE, don't set rendering intent in PostScript stream due to broken non-default CRDs.
useMaxVM	AT_PDF_BOOL	If TRUE, store all possible resources in VM.
lastWidth	AT_INT32	Used when setPageSize is TRUE to prevent unneeded setpagedevice calls.
lastHeight	AT_INT32	Used when setPageSize is TRUE to prevent unneeded setpagedevice calls.
bitmapResolution	AT_DWORD	DPI for bitmaps. Default is 300.
gradientResolution	AT_DWORD	DPI for gradients interior to the object (not edges). Can generally be lower than the bitmapResolution. Default is 150.
transparencyQuality	AT_DWORD	The transparency level. Range is 1-100.
ocContext	AT_DWORD	The optional-content context to use for visibility state information, or NULL to use the document's current states in the default context.
applyOCGPrintOverrides	AT_PDF_BOOL	When TRUE, apply print-specific visibility state settings from the optional-content group.
useFullResolutionJP2KData	AT_PDF_BOOL	Whether to use the maximum available JPEG 2000 resolution.
emitInRipSeps	AT_PDF_BOOL	When TRUE, requests that separations, one sheet per ink, be generated in the RIP (printer).
whichMarks	AT_DWORD	Page mark indication. A bit-wise OR of the enumIGPDFPageMarkFlags values.
westernMarksStyle	AT_PDF_BOOL	When TRUE, use western style for page marks.
doProofing	AT_PDF_BOOL	When TRUE, print using proofing settings.
proofProfile	char[256]	Description string for the proofing profile.
inkBlack	AT_PDF_BOOL	Proofing settings: simulate ink black.
paperWhite	AT_PDF_BOOL	Proofing settings: simulate paper white.
useExecForm	AT_PDF_BOOL	When TRUE, emit execform calls when emitting Form XObjects.
flattenInfo	LPAT_PDF_FLATTEN	A structure containing parameters that control tile

		flattening.
negative	AT_PDF_BOOL	When TRUE, invert the plate.
mirrorprint	CHAR	PostScript mirroring attribute. Currently not used.
numCollatedCopies	AT_DWORD	Enables collation for viewer.
emitFlatness	AT_PDF_BOOL	Set to TRUE to emit flatness.
trapType	AT_INT32	Specifies trap type.
TTasCIDT2	AT_PDF_BOOL	Set to TRUE to emit TrueType fonts as CIDType2 instead of as CIDFontType0.
markStyle	AT_DWORD	Specify the style to use for page marks.
lineWidth	AT_FLOAT	Line width to use for printer marks.
macQDPrinter	AT_PDF_BOOL	Set to TRUE if the printer is a Mac QuickDraw printer.
customMarksFileName	AT_UINT	If markStyle == -1, this should be a valid file name pointing to a valid .mrk file for custom printer marks.
pAGMPI	AT_VOID*	The AGMP interface pointer. Should be NULL.
disableFlattening	AT_PDF_BOOL	Disable flattening of the PDF file; transparency data will be ignored.
doNotDownloadFauxFonts	AT_PDF_BOOL	Allow user to select if faux files are downloaded.
suppressSnapToDevice	AT_PDF_BOOL	Allow user to control "Snap_To_Device".
suppressElement	AT_INT32	Unused. Set to 0.
maxFlatSeconds	AT_DWORD	Maximum flattener session seconds (of execution) before quality reduction.
testTilingMode	AT_DWORD	Provide a means for all 4 page rotations to be tiling-exercised at once.

Remarks:

All fields in the AT_PDF_PRINTPARAMS structure apply to PostScript file creation. AT_PDF_PRINTPARAMS are ignored when printing to non-PostScript devices.

1.3.3.5.36 AT_PDF_SECURITYDATA

This structure describes the data for the standard security handler.

Declaration:

```
typedef struct tagAT_PDF_SECURITYDATA
{
        AT_UINT size;
        AT_PDF_BOOL newUserPW;
        AT_PDF_BOOL hasUserPW;
        CHAR userPW[256];
        AT_PDF_BOOL newOwnerPW;
        AT_PDF_BOOL hasOwnerPW;
        CHAR ownerPW[256];
        AT_DWORD perms;
        LONG keyLength;
        AT_INT32 revision;
        AT_PDF_BOOL encryptMetadata;
        LONG encryptMethod;
        AT_PDF_BOOL encryptAttachmentsOnly;
        AT_INT32 version;
} AT_PDF_SECURITYDATA;
typedef AT_PDF_SECURITYDATA FAR* LPAT_PDF_SECURITYDATA;
```

Members:

Name	Type	Description
size	AT_UINT	Size of this structure.
newUserPW	AT_PDF_BOOL	TRUE if the user password should be changed.
hasUserPW	AT_PDF_BOOL	TRUE if there is a user password.
userPW	CHAR[256]	The user password string.
newOwnerPW	AT_PDF_BOOL	TRUE if the owner password should be changed; FALSE otherwise.
hasOwnerPW	AT_PDF_BOOL	TRUE if an owner password is provided; FALSE otherwise.
ownerPW	CHAR[256]	The owner password string.
perms	AT_DWORD	Permissions to allow. An OR of the enumIGPDFPermsFlags values.
keyLength	LONG	Encryption key length in byte.
revision	AT_INT32	Indicates /R value.
encryptMetadata	AT_PDF_BOOL	Flag that indicates whether document metadata will be encrypted.
encryptMethod	LONG	Method of encryption for filters to use. One of the enumIGPDFStdSecurityMethod values.
encryptAttachmentsOnly	AT_PDF_BOOL	Flag to indicate that only Attachments are encrypted - encryptMetadata and encryptAttachmentsOnly cannot both be true.
version	AT_INT32	Indicates a /V value.

Remarks:

The password strings in PDF are padded or truncated to exactly 32 bytes. If the password string is more than 32 bytes long, used only its first 32 bytes; if it is less than 32 bytes long, it padded by appending the required number of additional bytes from the beginning of the following padding string: <28 BF 4E 5E 4E 75 8A 41 64 00 4E 56 FF FA 01 08 2E 2E 00 B6 D0 68 3E 80 2F 0C A9 FE 64 53 69 7A >

1.3.3.5.37 AT_PDF_SYSFONT_PLATDATA

SysFont platform specific data.

Declaration:

```
typedef struct tagAT_PDF_SYSFONT_PLATDATA
{
        DWORD size;
        LPAT_VOID fontRef;

} AT_PDF_SYSFONT_PLATDATA;
typedef AT_PDF_SYSFONT_PLATDATA FAR* LPAT_PDF_SYSFONT_PLATDATA;
```

Members:

Name	Type	Description
size	DWORD	sizeof(AT_PDF_SYSFONT_PLATDATA).
fontRef	LPAT_VOID	The ATSFontRef of the sys font.

1.3.3.5.38 AT_PDF_TILE

Specifies printing flags.

Declaration:

```
typedef struct tagAT_PDF_TILE
{
     AT_DWORD overlap;
     AT_PDF_BOOL center;
     AT_DWORD marksflags;
     AT_DWORD paperWidth;
     AT_DWORD paperHeight;
     char* docTitle;
     char* docDate;
     char* docTime;
     AT_DWORD col;
     AT_DWORD row;
     AT_DWORD numCols;
     AT_DWORD numRows;
     AT_DWORD xOffset;
     AT_DWORD yOffset;

} AT_PDF_TILE;
typedef AT_PDF_TILE FAR* LPAT_PDF_TILE;
```

Members:

Name	Type	Description
overlap	AT_DWORD	Specifies the number of points to overlap (UI units may be different; application shall convert UI units to points).
center	AT_PDF_BOOL	Centers the pages' contents on the physical paper when set to TRUE.
marksflags	AT_DWORD	Specifies the printer marks to emit.
paperWidth	AT_DWORD	Specifies the width of the paper (in points); client-provided, since client has PPD access.
paperHeight	AT_DWORD	Specifies the height of the paper (in points); client-provided, since client has PPD access.
docTitle	char*	Specifies the title string for slug (optional).
docDate	char*	Specifies the date string for slug (optional).
docTime	char*	Specifies the time string for slug (optional).
col	AT_DWORD	Used for communicating the current page's state during print time: the current col (0 ... numcols-1).
row	AT_DWORD	Used for communicating the current page's state during print time: the current row.
numCols	AT_DWORD	Used for communicating the current page's state during print time: the numCols for this page.
numRows	AT_DWORD	Used for communicating the current page's state during print time: the numRows for this page.
xOffset	AT_DWORD	The amount to shift the first tile right, to center entire image on sheets.
yOffset	AT_DWORD	The amount to shift the first tile down, to center entire image on sheets.

1.3.3.5.39 AT_PDF_TILEEX

Specifies printing flags.

Declaration:

```
typedef struct tagAT_PDF_TILEEX
{
     AT_PDF_TILE pubRec;
     AT_DWORD imageablePaperWidth;
     AT_DWORD imageablePaperHeight;
     AT_DWORD unprintablePaperWidth;
     AT_DWORD unprintablePaperHeight;
     AT_DWORD indent;
     AT_INT32 rotateAngle;
     AT_UINT labelTemplate;
     AT_DOUBLE driverScale;
     AT_DOUBLE tileScale;

} AT_PDF_TILEEX;
typedef AT_PDF_TILEEX FAR* LPAT_PDF_TILEEX;
```

Members:

Name	Type	Description
pubRec	AT_PDF_TILE	Used for passing info to and from user.
imageablePaperWidth	AT_DWORD	Used internally.
imageablePaperHeight	AT_DWORD	Used internally.
unprintablePaperWidth	AT_DWORD	Used internally.
unprintablePaperHeight	AT_DWORD	Used internally.
indent	AT_DWORD	Used internally.
rotateAngle	AT_INT32	Used internally.
labelTemplate	AT_UINT	Used internally.
driverScale	AT_DOUBLE	Used internally.
tileScale	AT_DOUBLE	Used internally.

1.3.3.6 PDF Component Enumerations Reference

This section represents ImageGear PDF component enumerations including their meaning and values.

- enumIGPDEContentFlags
- enumIGPDEContentGetResourceFlags
- enumIGPDEElementCopyFlags
- enumIGPDEFontCreateFlags
- enumIGPDEFontCreateNeedFlags
- enumIGPDEFontProtection
- enumIGPDEGraphicStateWasSetFlags
- enumIGPDEImageAttrFlags
- enumIGPDEImageDataFlags
- enumIGPDEInsertElement
- enumIGPDEPathElementType
- enumIGPDEPathOpFlags
- enumIGPDEPSAttrFlags
- enumIGPDESoftMaskCreateFlags
- enumIGPDETextFlags
- enumIGPDEType
- enumIGPDEXGroupCreateFlags
- enumIGPDFBasicType
- enumIGPDFBookmarkFlags
- enumIGPDFCharset
- enumIGPDFCodePages
- enumIGPDFColorSpace
- enumIGPDFCompressions
- enumIGPDFDestinationType
- enumIGPDFDuplexEnum
- enumIGPDFFarEastFont
- enumIGPDFFixedValues
- enumIGPDFFlattenTilingMode
- enumIGPDFFontFlags
- enumIGPDFInclusion
- enumIGPDFInsertFlags
- enumIGPDFOCMDVisPolicy
- enumIGPDFPageDrawFlags
- enumIGPDFPageDrawMode
- enumIGPDFPageDrawSmoothFlags
- enumIGPDFPageMarkFlags
- enumIGPDFPageNumber
- enumIGPDFPageRange
- enumIGPDFPageTilingMode
- enumIGPDFPermReqObj
- enumIGPDFPermReqOpr
- enumIGPDFPermReqStatus
- enumIGPDFPermsFlags
- enumIGPDFPrintWhat
- enumIGPDFPrintWhatAnnot
- enumIGPDFRevision
- enumIGPDFRotation
- enumIGPDFSecurityInfoFlags
- enumIGPDFStdSecurityMethod
- enumIGPDFStreamType
- enumIGPDFSysFontMatchFlags
- enumIGPDFSysFontPackageType

- enumIGPDFWordFinderVersion
- enumIGPDFWordFlags

1.3.3.6.1 enumIGPDEContentFlags

Bit field for AT_PDE_CONTENTATTRS.

Values:

IG_PDE_SET_CACHE_DEVICE	0x0001	If set, cacheDevice contains 6 cache device values.
IG_PDE_SET_CHAR_WIDTH	0x0002	If set, cacheDevice contains 2 charwidth values.
IG_PDE_FORM_MATRIX	0x0004	If set, formMatrix contains a valid matrix.

1.3.3.6.2 enumIGPDEContentGetResourceFlags

Bit field for AT_PDE_CONTENTATTRS.

Values:

IG_PDE_GET_FONTS	0	Obtain font resources.
IG_PDE_GET_XOBJECTS	1	Obtain Xobject resources.
IG_PDE_GET_COLORSPACES	2	Obtain color space resources.

1.3.3.6.3 enumIGPDEElementCopyFlags

Bitfield for PDE element copy.

Values:

IG_PDE_ELEMENT_COPY_FOR_CLIP	0x0001	Copied element does not need gstate or clip.
IG_PDE_ELEMENT_COPY_CLIPPING	0x0002	Acquire the clip path and put it in the copied object.

1.3.3.6.4 enumIGPDEFontCreateFlags

Flags for PDE font creation routine. If you want to subset a font, set both the IG_PDE_FONT_CREATE_EMBEDDED and IG_PDE_FONT_WILL_SUBSET flags.

Values:

IG_PDE_FONT_CREATE_NOT_ALLOWED	-1	Creation is not allowed. Usually returns by SysFont's "getCreateFlags" when the combination of SysFont and SysEncoding is not allowed.
IG_PDE_FONT_CREATE_EMBEDDED	0x0001	Embed the font. Create an embedded font. By itself, this will not subset the font.
IG_PDE_FONT_WILL_SUBSET	0x0002	Subset the font. If you want to subset a font, set both the IG_PDE_FONT_CREATE_EMBEDDED and IG_PDE_FONT_WILL_SUBSET flags. You must call "subsetNow" to actually subset the font. Both embedding and sub-setting a font creates a CFF font.
IG_PDE_FONT_DO_NOT_EMBED	0x0004	Do not embed the font. You cannot set both this and the IG_PDE_FONT_WILL_SUBSET flags. Nor can you set IG_PDE_FONT_CREATE_EMBEDDED.
IG_PDE_FONT_ENCODE_BY_GID	0x0008	Create a CIDFont with identity (GID) encoding.
IG_PDE_FONT_DEFER_WIDTHS	0x0010	Wait to get widths until later (affects Type0 fonts only).
IG_PDE_FONT_CREATE_SUBSET	0x0002	Subset the font. If you want to subset a font, set both the IG_PDE_FONT_CREATE_EMBEDDED and IG_PDE_FONT_WILL_SUBSET flags. You must call "subsetNow" to actually subset the font. Both embedding and sub-setting a font creates a CFF font.
IG_PDE_FONT_CREATE_GID_OVERRIDE	0x0020	The library will convert cp to gid with identity embedded.
IG_PDE_FONT_CREATE_TO_UNICODE	0x0040	Create ToUnicode cmap.
IG_PDE_FONT_CREATE_ALL_WIDTHS	0x0080	Supply entire widths table (affects Type0 fonts only).

1.3.3.6.5 enumIGPDEFontCreateNeedFlags

Flags for PDE Font CreateNeedFlags.

Values:

IG_PDE_FONT_CREATE_NEED_WIDTHS	0x00010000	Need to create width.
IG_PDE_FONT_CREATE_NEED_TO_UNICODE	0x00020000	Need to create ToUnicode stream.
IG_PDE_FONT_CREATE_NEED_EMBED	0x00040000	Need to embed it.

1.3.3.6.6 enumIGPDEFontProtection

Setting for disabling font embedding.

Values:

IG_PDE_FONT_NO_EMBEDDING	0x00000001	Flags for protection of AT_PDE_FONTATTRS - embedding is not allowed.
IG_PDE_FONT_NO_EDITABLE_EMBEDDING	0x00000002	Flags for protection of AT_PDE_FONTATTRS - editable embedding is not allowed.

1.3.3.6.7 enumIGPDEGraphicStateWasSetFlags

Structure describing the graphics state that was set.

Values:

IG_PDE_FILL_CSPACE_WAS_SET	0x0001	A fill color space was set, corresponding to the cs (setcolorspace) operator.
IG_PDE_FILL_CVALUE_WAS_SET	0x0002	A color fill value was set, corresponding to the sc (setcolor) operator.
IG_PDE_STROKE_CSPACE_WAS_SET	0x0004	A color space stroke value was set, corresponding to the CS (setcolorspace) operator.
IG_PDE_STROKE_CVALUE_WAS_SET	0x0008	A color stroke value was set, corresponding to the SC (setcolor) operator.
IG_PDE_DASH_WAS_SET	0x0010	A dash specification was set, corresponding to the d (setdash) operator.
IG_PDE_LINE_WIDTH_WAS_SET	0x0020	The line width was set, corresponding to the w (setlinewidth) operator.
IG_PDE_MITER_LIMIT_WAS_SET	0x0040	The miter limit was set, corresponding to the M (setmiterlimit) operator.
IG_PDE_FLATNESS_WAS_SET	0x0080	Line flatness was set, corresponding to the i (setflat) operator.
IG_PDE_LINE_CAP_WAS_SET	0x0100	Line cap style was set, corresponding to the J (setlinecap) operator.
IG_PDE_LINE_JOIN_WAS_SET	0x0200	Line join style was set, corresponding to the j (setlinejoin) operator.
IG_PDE_RENDER_INTENT_WAS_SET	0x0400	A color rendering intent was set, corresponding to the Intent key in the image dictionary.
IG_PDE_EXT_GSTATE_WAS_SET	0x0800	An extended graphics state was set, corresponding to the gs operator.

1.3.3.6.8 enumIGPDEImageAttrFlags

Flags for AT_PDE_IMAGEATTRS. See Section 4.8.4 in the PDF Reference for more information on image attributes.

Values:

IG_PDE_IMAGE_EXTERNAL	0x0001	Image is an XObject.
IG_PDE_IMAGE_MASK	0x0002	Image is an imagemask.
IG_PDE_IMAGE_INTERPOLATE	0x0004	Interpolate is true.
IG_PDE_IMAGE_HAVE_DECODE	0x0008	We have a decode array.
IG_PDE_IMAGE_INDEXED	0x0010	Uses an indexed color space.
IG_PDE_IMAGE_MASKED_BY_POSITION	0x0020	Image has a Mask key containing an ImageMask stream.
IG_PDE_IMAGE_MASKED_BY_COLOR	0x0040	Image has a Mask key containing an array of color values.

1.3.3.6.9 enumIGPDEImageDataFlags

Image Data Flags.

Values:

AT_PDE_IMAGE_DATA_NOT_ENCODED	0x0000	Indicates filter is active; data is not encoded.
AT_PDE_IMAGE_ENCODED_DATA	0x0001	Indicates filter is active; data is encoded.

1.3.3.6.10 enumIGPDEInsertElement

Used for inserting a PDE element into the content.

Values:

IG_PDE_BEFORE_FIRST	((LONG) -1)	Specifies position before the first element. Usually used to insert first content element.
IG_PDE_AFTER_LAST	(IG_PDF_FIXED_MAX - 1)	Specifies the last element position. Usually used to insert last content element.

1.3.3.6.11 enumIGPDEPathElementType

Constant values that describe path segment operators in PDE path elements.

Values:

IG_PDE_MOVE_TO	0	Designates m (moveto) operator, which moves the current point.
IG_PDE_LINE_TO	1	Designates l (lineto) operator, which appends a straight line segment from the current point.
IG_PDE_CURVE_TO	2	Designates c (curveto) operator, which appends a Bezier curve to the path.
IG_PDE_CURVE_TO_V	3	Designates v (curveto) operator, which appends a Bezier curve to the current path when the first control point coincides with initial point on the curve.
IG_PDE_CURVE_TO_Y	4	Designates y (curveto) operator, which appends a Bezier curve to the current path when the second control point coincides with final point on the curve.
IG_PDE_RECT	5	Designates re operator, which adds a rectangle to the current path.
IG_PDE_CLOSE_PATH	6	Designates h (closepath) operator, which closes the current subpath.

1.3.3.6.12 enumIGPDEPathOpFlags

Flags for paint operators in a PDE path.

Values:

IG_PDE_INVISIBLE	0x00	Path is neither stroked nor filled, so it is invisible.
IG_PDE_STROKE	0x01	Stroke the path, as with the S (stroke) operator.
IG_PDE_FILL	0x02	Fills the path, using the nonzero winding number rule to determine the region to fill, as with the f (fill) operator.
IG_PDE_EO_FILL	0x04	Fills the path, using the even-odd rule to determine the region to fill, as with the f* (eofill) operator.

1.3.3.6.13 enumIGPDEPSAttrFlags

Flags for AT_PDE_PSATTRS.

Values:

IG_PDE_PS_INLINE	0	Inline PostScript.

1.3.3.6.14 enumIGPDESoftMaskCreateFlags

Flags for use with SoftMask create.

Values:

IG_PDE_SOFTMASK_TYPE_LUMINOSITY	0x0001	Specifies how the mask is to be computed.
IG_PDE_SOFTMASK_TYPE_ALPHA	0x0002	Specifies how the mask is to be computed.

1.3.3.6.15 enumIGPDETextFlags

Flags used in the text API.

Values:

IG_PDE_TEXT_RUN	0x0001	Specifies an action for the run.
IG_PDE_TEXT_CHAR	0x0002	Specifies an action for single character.
IG_PDE_TEXT_PAGE_SPACE	0x0004	Specifies user space.
IG_PDE_TEXT_GET_BOUNDS	0x0008	Specifies using the font descriptor's FontBBox.

1.3.3.6.16 enumIGPDEType

Types of the editing objects.

Values:

IG_PDE_CONTENT	0	Content.
IG_PDE_TEXT	1	Text.
IG_PDE_PATH	2	Path.
IG_PDE_IMAGE	3	Image.
IG_PDE_FORM	4	Form.
IG_PDE_POSTSCRIPT	5	PostScript.
IG_PDE_XOBJECT	6	XObject.
IG_PDE_CLIP	7	Clip.
IG_PDE_FONT	8	Font.
IG_PDE_COLORSPACE	9	ColorSpace.
IG_PDE_GSTATE	10	Graphic State.
IG_PDE_PLACE	11	Place.
IG_PDE_CONTAINER	12	Container.
IG_PDF_SYSFONT	13	System Font.
IG_PDE_PATTERN	14	Pattern.
IG_PDE_DEVICENCOLORS	15	Device N Colors.
IG_PDE_SHADING	16	Shading.
IG_PDE_GROUP	17	Group.
IG_PDE_UNKNOWN	18	Unknown.
IG_PDE_BEGIN_CONTAINER	19	Begin Container.
IG_PDE_END_CONTAINER	20	End Container.
IG_PDE_BEGIN_GROUP	21	Begin Group.
IG_PDE_END_GROUP	22	End Group.
IG_PDE_XGROUP	23	XGroup.
IG_PDE_SOFTMASK	24	SoftMask.
IG_PDF_SYSENCODING	25	System Encoding.
IG_PDE_DOC	26	Document.
IG_PDE_PAGE	27	Page.
IG_PDE_READER	28	Reader.
IG_PDE_WRITER	29	Writer.
IG_PDE_TEXTITEM	30	Text Item.
IG_PDE_LASTTYPE	31	Last Type.

1.3.3.6.17 enumIGPDEXGroupCreateFlags

Enumerated data type used to specify the type of transparency group to create.

Values:

IG_PDE_XGROUP_TYPE_TRANSPARENCY	0x0001	Creates a transparency XGroup object.

1.3.3.6.18 enumIGPDFBasicType

Basic PDF objects.

Values:

IG_PDF_BASIC_NULL	0	Null object.
IG_PDF_BASIC_INT	1	Integer object.
IG_PDF_BASIC_FIXED	2	Fixed (real) object.
IG_PDF_BASIC_BOOL	3	Boolean object.
IG_PDF_BASIC_NAME	4	Name object.
IG_PDF_BASIC_STRING	5	String object.
IG_PDF_BASIC_DICT	6	Dictionary object.
IG_PDF_BASIC_ARRAY	7	Array object.
IG_PDF_BASIC_STREAM	8	Stream object.

1.3.3.6.19 enumIGPDFBookmarkFlags

Represents PDF bookmark flags.

Values:

IG_PDF_BOOKMARK_FONT_ITALIC	1	Italic font.
IG_PDF_BOOKMARK_FONT_BOLD	2	Bold font.

1.3.3.6.20 enumIGPDFCharset

Represents the PDF Character Set.

Values:

IG_PDF_CHARSET_UNKNOWN	0	The font does not use Adobe Standard Encoding.
IG_PDF_CHARSET_ROMAN	1	The font uses Adobe Standard encoding. This is determined by the "Uses Adobe Standard Encoding" bit in the font descriptor.
IG_PDF_CHARSET_EXPERT	2	Currently unused.
IG_PDF_CHARSET_LAST	3	Placeholder for the last value of this enumeration.

1.3.3.6.21 enumIGPDFCodePages

Code page character-mapping constants.

Values:

IG_PDF_CODEPAGE_WIN_EAST_EUROPEAN_ROMAN	1250	Windows code pages.
IG_PDF_CODEPAGE_WIN_CYRILLIC	1251	Windows code pages.
IG_PDF_CODEPAGE_WIN_GREEK	1253	Windows code pages.
IG_PDF_CODEPAGE_WIN_TURKISH	1254	Windows code pages.
IG_PDF_CODEPAGE_WIN_HEBREW	1255	Windows code pages.
IG_PDF_CODEPAGE_WIN_ARABIC	1256	Windows code pages.
IG_PDF_CODEPAGE_WIN_BALTIC	1257	Windows code pages.
IG_PDF_CODEPAGE_MAC_CENTRAL_EUROPEAN	-9994	Macintosh pseudo code pages.
IG_PDF_CODEPAGE_MAC_CROATIAN	-9993	Macintosh pseudo code pages.
IG_PDF_CODEPAGE_MAC_ROMANIAN	-9992	Macintosh pseudo code pages.
IG_PDF_CODEPAGE_MAC_CYRILLIC	-9991	Macintosh pseudo code pages.
IG_PDF_CODEPAGE_MAC_UKRAINIAN	-9990	Macintosh pseudo code pages.
IG_PDF_CODEPAGE_MAC_GREEK	-9989	Macintosh pseudo code pages.
IG_PDF_CODEPAGE_MAC_TURKISH	-9988	Macintosh pseudo code pages.
IG_PDF_CODEPAGE_MAC_HEBREW	-9987	Macintosh pseudo code pages.
IG_PDF_CODEPAGE_MAC_ARABIC	-9986	Macintosh pseudo code pages.

1.3.3.6.22 enumIGPDFColorSpace

Specifies the color space in which a color value is specified (for example, RGB or Grayscale).

Values:

IG_PDF_DEVICE_GRAY	Grayscale color specification. Requires 1 value entry to specify the color.
IG_PDF_DEVICE_RGB	Red-Green-Blue color specification. Requires 3 value entries to specify the color.
IG_PDF_DEVICE_CMYK	Cyan-Magenta-Yellow-Black color specification. Requires 4 value entries to specify the color.

1.3.3.6.23 enumIGPDFCompressions

PDF compressions for raster images.

Values:

IG_PDF_COMPRESSION_NONE	0	No compression.
IG_PDF_COMPRESSION_CCITT_G3	1	CCITT G3 compression.
IG_PDF_COMPRESSION_CCITT_G4	2	CCITT G4 compression.
IG_PDF_COMPRESSION_CCITT_G32D	3	CCITT G32D compression.
IG_PDF_COMPRESSION_JPEG	4	JPEG compression.
IG_PDF_COMPRESSION_LZW	5	LZW compression.
IG_PDF_COMPRESSION_RLE	6	RLE compression.
IG_PDF_COMPRESSION_DEFLATE	7	Deflate compression.

1.3.3.6.24 enumIGPDFDestinationType

Represents the types of PDF object destination.

Values:

IG_PDF_DEST_INVALID	0	Invalid destination.
IG_PDF_DEST_EXPLICIT	1	Explicit destination.
IG_PDF_DEST_NAMED	2	Named destination.

1.3.3.6.25 enumIGPDFDuplexEnum

Specifies duplex values.

Values:

IG_PDF_DUPLEX_OFF	0x0000	Respect whatever duplex option was selected in printer preferences.
IG_PDF_DUPLEX_ON_TUMBLE_SHORT	0x0001	Specify Duplex mode, tumbling on the short axis of the page (e.g., tablet-style).
IG_PDF_DUPLEX_ON_TUMBLE_LONG	0x0002	Specify Duplex mode, tumbling on the long axis of the page (e.g., Portrait book-style).
IG_PDF_DUPLEX_FORCE_SIMPLEX	0x0003	Force simplex printing, ignoring the printer preferences.

1.3.3.6.26 enumIGPDFFarEastFont

Specifies CJK font related option for PostScript printing.

Values:

IG_PDF_FAREASTFONT_DOWNLOAD_ALL	Download all CJK fonts to printer.
IG_PDF_FAREASTFONT_DOWNLOAD_NONE	Download only embedded fonts to printer.
IG_PDF_FAREASTFONT_PRINT_AS_IMAGE	Do not download CJK fonts to printer. Render characters and send them to printer as bitmaps. PS Level 1 should use this to print CJK.

1.3.3.6.27 enumIGPDFFixedValues

PDF Fixed value. A variety of predefined fixed-point constants.

Values:

IG_PDF_FIXED_ZERO	((LONG) 0x00000000L)	0
IG_PDF_FIXED_HUNDREDTH	((LONG) 0x0000028FL)	1/100
IG_PDF_FIXED_SIXTEENTH	((LONG) 0x00001000L)	1/16
IG_PDF_FIXED_TWELFTH	((LONG) 0x00001555L)	1/12
IG_PDF_FIXED_TENTH	((LONG) 0x00001999L)	1/10
IG_PDF_FIXED_EIGHTH	((LONG) 0x00002000L)	1/8
IG_PDF_FIXED_QUARTER	((LONG) 0x00004000L)	1/4
IG_PDF_FIXED_THIRD	((LONG) 0x00005555L)	1/3
IG_PDF_FIXED_HALF	((LONG) 0x00008000L)	1/2
IG_PDF_FIXED_TWOTHIRDS	((LONG) 0x0000AAAAL)	2/3
IG_PDF_FIXED_THREEQUARTERS	((LONG) 0x0000C000L)	3/4
IG_PDF_FIXED_PI4	((LONG) 0x0000c910L)	PI/4
IG_PDF_FIXED_SEVENEIGHTS	((LONG) 0x0000E000L)	7/8
IG_PDF_FIXED_ONE1	((LONG) 0x0000ffffL)	-1
IG_PDF_FIXED_ONE	((LONG) 0x00010000L)	1
IG_PDF_FIXED_PI2	((LONG) 0x00019220L)	PI/2
IG_PDF_FIXED_GOLDEN	((LONG) 0x00019e37L)	Golden.
IG_PDF_FIXED_TEN	((LONG) 0x000A0000L)	10
IG_PDF_FIXED_MAX	((LONG) 0x7FFFFFFF)	Max fixed-point value.
IG_PDF_FIXED_MIN	((LONG) 0x80000000)	Min fixed-point value.
IG_PDF_FIXED_NEGATIVE_INFINITY	((LONG) IG_PDF_FIXED_MAX)	Negative fixed-point infinity.
IG_PDF_FIXED_POSITIVE_INFINITY	((LONG) IG_PDF_FIXED_MIN)	Positive fixed-point infinity.

1.3.3.6.28 enumIGPDFFlattenTilingMode

Specifies tiled flattening modes.

Values:

IG_PDF_NO_TILING	No tiling.
IG_PDF_CONSTANT_TILING	Constant tiling.
IG_PDF_ADAPTIVE_TILING	Adaptive tiling.

1.3.3.6.29 enumIGPDFFontFlags

Font flags. Constants that indicate a font's attributes (fixed width, roman or symbolic, sans serif, and so forth).

Values:

IG_PDF_FIXED_WIDTH	0x00000001	All glyphs in the font are the same width.
IG_PDF_SERIF	0x00000002	The font is a serif font.
IG_PDF_PI	0x00000004	The font is a symbolic (pi) font.
IG_PDF_SCRIPT	0x00000008	The font is a script font.
IG_PDF_STD_ENCODING	0x00000020	The font uses standard encoding.
IG_PDF_ITALIC	0x00000040	The font is an italic font.
IG_PDF_ALL_CAP	0x00010000	The font is an all-caps font.
IG_PDF_SMALL_CAP	0x00020000	The font is a small caps font.
IG_PDF_FORCE_BOLD	0x00040000	Force bold characters to draw bold even at small point sizes.

1.3.3.6.30 enumIGPDFInclusion

This enumeration specifies how to include a resource in a file.

Values:

IG_PDF_INCLUDE_ONCE_PER_DOC	Include the resource only once per file.
IG_PDF_INCLUDE_ON_EVERY_PAGE	Include the resource on every page in the file.
IG_PDF_INCLUDE_NEVER	Never include the resource.
IG_PDF_INCLUDE_WHEN_NEEDED	Include the resources only when needed.
IG_PDF_INCLUDE_BY_RANGE	Include the range of resource.

1.3.3.6.31 enumIGPDFInsertFlags

Specifies PDF page insert flags.

Values:

IG_PDF_INSERT_BOOKMARKS	Insert bookmarks only.
IG_PDF_INSERT_ALL	Insert all.
IG_PDF_INSERT_THREADS	Insert threads only.

1.3.3.6.32 enumIGPDFOCMDVisPolicy

Represents the 4 legal values for the /P key in an OCMD dictionary. They specify the visibility of content with respect to the on/off state of the OCGs layers listed in the OCMD dictionary.

Values:

IGPDFOCMDVisibility_AllOn	Content in the member groups is visible only when all groups are ON.
IGPDFOCMDVisibility_AnyOn	Content in the member groups is visible only when any of the groups is ON.
IGPDFOCMDVisibility_AnyOff	Content in the member groups is visible only when any of the groups is OFF.
IGPDFOCMDVisibility_AllOff	Content in the member groups is visible only when all groups are OFF.

1.3.3.6.33 enumIGPDFPageDrawFlags

Bit flags indicating how a page is rendered.

Values:

IG_PDF_PAGE_DO_LAZY_ERASE	Erase the page while rendering only as needed.
IG_PDF_PAGE_USE_ANNOT_FACES	Draw annotation appearances.
IG_PDF_PAGE_IS_PRINTING	The page is being printed.

1.3.3.6.34 enumIGPDFPageDrawMode

Specifies PDF page drawing mode.

Values:

IG_PDF_PAGE_DRAW_ENTIRE_PAGE	Render entire page content.
IG_PDF_PAGE_DRAW_VISIBLE_AREA	Render visible page area.

1.3.3.6.35 enumIGPDFPageDrawSmoothFlags

Specifies bit flags indicating how a page is rendered.

Values:

IG_PDF_PAGE_DRAW_SMOOTH_TEXT	0x0001	Draw smooth text.
IG_PDF_PAGE_DRAW_SMOOTH_LINE_ART	0x0002	Draw smooth line art.
IG_PDF_PAGE_DRAW_SMOOTH_IMAGE	0x0004	Draw smooth image.
IG_PDF_ENHANCE_THIN_LINES	0x0008	Enhance thin lines.

1.3.3.6.36 enumIGPDFPageMarkFlags

Bit flags indicating which page marks are emitted for color separations.

Values:

IG_PDF_PAGE_EMIT_COLOR_BARS	Emit color bars.
IG_PDF_PAGE_EMIT_REG_MARKS	Emit register marks.
IG_PDF_PAGE_EMIT_CROP_MARKS	Emit crop marks.
IG_PDF_PAGE_EMIT_BLEED_MARKS	Emit bleed marks.
IG_PDF_PAGE_EMIT_PAGE_INFO	Emit page info.
IG_PDF_PAGE_EMIT_TRIM_MARKS	Emit trim marks.
IG_PDF_PAGE_EMIT_SLUR_MARKS	Emit slur marks.

1.3.3.6.37 enumIGPDFPageNumber

PageNumber specification.

Values:

IG_PDF_BEFORE_FIRST_PAGE	-1	Specifies position before the first page. Usually used to insert first document page.
IG_PDF_LAST_PAGE	-2	Specifies the last page position. Usually used to insert last document page.

1.3.3.6.38 enumIGPDFPageRange

The types of PDF pages range.

Values:

IG_PDF_ALL_PAGES	-3 (&HFFFFFFFD)	All pages.
IG_PDF_EVEN_PAGES	-5 (&HFFFFFFFB)	Even pages only.
IG_PDF_ODD_PAGES	-4 (&HFFFFFFFC)	Odd pages only.

1.3.3.6.39 enumIGPDFPageTilingMode

Specifies PDF page tiling mode.

Values:

IG_PDF_NO_PAGE_TILING	Print all pages normally.
IG_PDF_TILE_ALL_PAGES	Use tiling settings for all pages.
IG_PDF_TILE_LARGE_PAGES	Use tiling only for pages larger than size indicated in PDTileRec of tileInfo.

1.3.3.6.40 enumIGPDFPermReqObj

Enumerated data type used to describe the target object of a permission request.

Values:

IG_PDF_PERM_REQ_OBJ_DOC	1	Document.
IG_PDF_PERM_REQ_OBJ_PAGE	2	Page.
IG_PDF_PERM_REQ_OBJ_LINK	3	Link.
IG_PDF_PERM_REQ_OBJ_BOOKMARK	4	Bookmark.
IG_PDF_PERM_REQ_OBJ_THUMBNAIL	5	Thumbnail.
IG_PDF_PERM_REQ_OBJ_ANNOT	6	Annotation.
IG_PDF_PERM_REQ_OBJ_FORM	7	Form.
IG_PDF_PERM_REQ_OBJ_SIGNATURE	8	Signature.
IG_PDF_PERM_REQ_OBJ_LAST	9	Used for checking cache size.

1.3.3.6.41 enumIGPDFPermReqOpr

Enumerated data type used to describe the target operation of a permissions request.

Values:

IG_PDF_PERM_REQ_OPR_ALL	1	Check all operations.
IG_PDF_PERM_REQ_OPR_CREATE	2	Generic operation.
IG_PDF_PERM_REQ_OPR_DELETE	3	Delete.
IG_PDF_PERM_REQ_OPR_MODIFY	4	Modify.
IG_PDF_PERM_REQ_OPR_COPY	5	Copy.
IG_PDF_PERM_REQ_OPR_ACCESSIBLE	6	For Accessibility use
IG_PDF_PERM_REQ_OPR_SELECT	7	For doc or page, selecting (not copying) text or graphics.
IG_PDF_PERM_REQ_OPR_OPEN	8	For document open.
IG_PDF_PERM_REQ_OPR_SECURE	9	For doc to changing security settings.
IG_PDF_PERM_REQ_OPR_PRINT_HIGH	10	For doc, Regular printing.
IG_PDF_PERM_REQ_OPR_PRINT_LOW	11	For doc, low quality printing.
IG_PDF_PERM_REQ_OPR_FILL_IN	12	Form fill-in or Sign existing field.
IG_PDF_PERM_REQ_OPR_ROTATE	13	Rotate.
IG_PDF_PERM_REQ_OPR_CROP	14	Crop.
IG_PDF_PERM_REQ_OPR_SUMMARIZE	15	For summarize notes.
IG_PDF_PERM_REQ_OPR_INSERT	16	Insert.
IG_PDF_PERM_REQ_OPR_REPLACE	17	For page.
IG_PDF_PERM_REQ_OPR_REORDER	18	For page.
IG_PDF_PERM_REQ_OPR_FULL_SAVE	19	For doc.
IG_PDF_PERM_REQ_OPR_IMPORT	20	For notes & Image.
IG_PDF_PERM_REQ_OPR_EXPORT	21	For notes. ExportPS should check print.
IG_PDF_PERM_REQ_OPR_ANY	22	Used for checking to see if any operation is allowed.
IG_PDF_PERM_REQ_OPR_UNKNOWNOPR	23	Used for error checking.
IG_PDF_PERM_REQ_OPR_SUBMIT_STANDALONE	24	Submit forms outside of the browser.
IG_PDF_PERM_REQ_OPR_SPAWN_TEMPLATE	25	Allows form to spawn template page.
IG_PDF_PERM_REQ_OPR_LAST	26	This should be always the last item.

1.3.3.6.42 enumIGPDFPermReqStatus

An enumerated data type that provides the status of PDF Doc-related permissions methods.

Values:

IG_PDF_PERM_REQ_DENIED	1	Request was denied.
IG_PDF_PERM_REQ_GRANTED	0	Request was granted.
IG_PDF_PERM_REQ_UNKNOWN_OBJECT	1	The object is unknown.
IG_PDF_PERM_REQ_UNKNOWN_OPERATION	2	The operation is unknown.
IG_PDF_PERM_REQ_OPERATION_NA	3	The operation is not applicable for the specified object.
IG_PDF_PERM_REQ_PENDING	4	The handler doesn't have enough info to answer at this point. Try again later.

1.3.3.6.43 enumIGPDFPermsFlags

Flags that describe permissions wanted and granted for a document. Not all permissions will be granted if the document is protected or if the document is newer version than the application knows about.

Values:

IG_PDF_PERM_OPEN	0x01	The user is permitted to open and decrypt the document.
IG_PDF_PERM_SECURE	0x02	The user is permitted to change the document's security settings.
IG_PDF_PERM_PRINT	0x04	The user is permitted to print the document. Page Setup access is unaffected by this permission, since that affects Acrobat's preferences - not the document's. In the Document Security dialog, this corresponds to the Printing entry.
IG_PDF_PERM_EDIT	0x08	The user is permitted to edit the document more than adding or modifying text notes (see also IG_PDF_PERM_EDIT_NOTES). In the Document Security dialog, this corresponds to the Changing the Document entry.
IG_PDF_PERM_COPY	0x10	The user is permitted to copy information from the document to the clipboard. In the Document Security dialog, this corresponds to the Content Copying or Extraction entry.
IG_PDF_PERM_EDIT_NOTES	0x20	The user is permitted to add, modify, and delete text notes (see also IG_PDF_PERM_EDIT). In the Document Security dialog, this corresponds to the Authoring Comments and Form Fields entry.
IG_PDF_PERM_SAVE_AS	0x40	The user is permitted to perform a "Save As..." If both IG_PDF_PERM_EDIT and IG_PDF_PERM_EDIT_NOTES are disallowed, "Save" will be disabled but "Save As..." is enabled. The "Save As..." menu item is not necessarily disabled even if the user is not permitted to perform a "Save As...". Not settable by clients.
IG_PDF_PERM_EXT	0x80	
IG_PDF_PRIV_PERM_FILL_AND_SIGN	0x100	Override other enumIGPDFPermsFlags bits. It allows a user to fill-in or sign existing form or signature fields.
IG_PDF_PRIV_PERM_ACCESSIBLE	0x200	Override IG_PDF_PERM_COPY to enable Accessibility API. If a document is saved in Rev2 format (Acrobat 4.0 compatible), only IG_PDF_PERM_COPY bit is checked to determine Accessibility API state.
IG_PDF_PRIV_PERM_DOC_ASSEMBLY	0x400	Override various IG_PDF_PERM_EDIT bit and allow the following operations; page insert/delete/rotate and create

		bookmark and thumbnail.
IG_PDF_PRIV_PERM_HIGH_PRINT	0x800	This bit is supplement to IG_PDF_PERM_PRINT. If it is clear (disabled) only low quality printing (Print As Image) is allowed. Under UNIX platforms, where "Print As Image" doesn't exist, printing is disabled.
IG_PDF_PERM_OWNER	0x8000	The user is permitted to perform all operations, regardless of the permissions specified by the document. Unless this permission is set, the document's permissions will be reset to those in the document after a full save.
IG_PDF_PRIV_PERM_FORM_SUBMIT	0x10000	Should be set if user can submit forms outside of the browser. This bit is supplement to IG_PDF_PRIV_PERM_FILL_AND_SIGN.
IG_PDF_PRIV_PERM_FORM_SPAWN_TEMPL	0x20000	Should be set if user can spawn template pages. This bit will allow page template spawning even if IG_PDF_PERM_EDIT and IG_PDF_PERM_EDIT_NOTES are clear.
IG_PDF_PERM_ALL	0xFFFFFFFF	Sets all permissions, including bit-fields that are reserved for future use.
IG_PDF_PERM_SETTABLE	IG_PDF_PERM_PRINT + IG_PDF_PERM_EDIT + IG_PDF_PERM_COPY + IG_PDF_PERM_EDIT_NOTES	The OR of all operations that can be set by the user in the Standard Security dialog (IG_PDF_PERM_PRINT + IG_PDF_PERM_EDIT + IG_PDF_PERM_COPY + IG_PDF_PERM_EDIT_NOTES)
IG_PDF_PERM_USER	IG_PDF_PERM_ALL - IG_PDF_PERM_OPEN - IG_PDF_PERM_SECURE	All permissions.

1.3.3.6.44 enumIGPDFPrintWhat

Specifies the kind of data to be printed, e.g., only the document, document and comments, etc.

Values:

IG_PDF_PRINT_DOCUMENT	Print only the document.
IG_PDF_PRINT_DOCUMENT_AND_COMMENTS	Print the document and associated annotations.
IG_PDF_PRINT_FORM_FIELDS_ONLY	Print only the data within form fields.
IG_PDF_PRINT_COUNT	Service value used to mark end last enum value.
IG_PDF_PRINT_MIN	Service value specifies minimum constant of this enum.

1.3.3.6.45 enumIGPDFPrintWhatAnnot

Specifies which extra annotations to print.

Values:

IG_PDF_PRINT_NO_EXTRAS	No extra printing marks.
IG_PDF_PRINT_TRAP_ANNOTS	Print trap annotations.
IG_PDF_PRINT_PRINTER_MARKS	Print printer marks.

1.3.3.6.46 enumIGPDFRevision

Specifies /R revision value.

Values:

IG_PDF_REVISION_2	2	Support by Acrobat 3.0 and up.
IG_PDF_REVISION_3	3	Support by Acrobat 5.0 and up.
IG_PDF_REVISION_4	4	Support by Acrobat 6.0 and up.

1.3.3.6.47 enumIGPDFRotation

Specifies page rotation, in degrees. Used for routines that set/get the value of a page's Rotate key.

Values:

IG_PDF_ROTATE_0	Zero rotate angle.
IG_PDF_ROTATE_90	Rotate angle is 90 degrees.
IG_PDF_ROTATE_180	Rotate angle is 180 degrees.
IG_PDF_ROTATE_270	Rotate angle is 270 degrees.

1.3.3.6.48 enumIGPDFSecurityInfoFlags

Flags used to specify various information about the Acrobat viewer's security and permissions.

Values:

IG_PDF_INFO_HAS_USER_PW	IG_PDF_PERM_OPEN	The document has a user password.
IG_PDF_INFO_HAS_OWNER_PW	IG_PDF_PERM_SECURE	The document has an owner password.
IG_PDF_INFO_CAN_PRINT	IG_PDF_PERM_PRINT	The document can be printed.
IG_PDF_INFO_CAN_EDIT	IG_PDF_PERM_EDIT	The document can be modified, for example by adding notes, links, or bookmarks.
IG_PDF_INFO_CAN_COPY	IG_PDF_PERM_COPY	The document text and graphics can be copied to the clipboard.

1.3.3.6.49 enumIGPDFStdSecurityMethod

Specifies standard security algorithms.

Values:

IG_PDF_STD_SECURITY_METHOD_RC4_V2	2	RC4 algorithm for encryption.
IG_PDF_STD_SECURITY_METHOD_AES_V1	5	AES algorithm for encryption with a zero initialized iv.
IG_PDF_STD_SECURITY_METHOD_AES_V2	6	AES algorithm for encryption with a random initialized iv.
IG_PDF_STD_SECURITY_METHOD_AES_V3	7	AES algorithm for encryption with a 4 byte random iv.

1.3.3.6.50 enumIGPDFStreamType

Specifies type of the registered stream event, used in IGPDFCtl.RegisterStreamEvent.

Values:

IG_STREAM_READ	0	Used to register StreamRead event for read-only from Stream.
IG_STREAM_WRITE	1	Used to register StreamWrite event for writing to Stream.

1.3.3.6.51 enumIGPDFSysFontMatchFlags

Font matching flags for SysFont find routine.

Values:

IG_PDF_SYSFONT_MATCH_NAME_AND_CHARSET	0x0001	Match the font name and character set.
IG_PDF_SYSFONT_MATCH_FONT_TYPE	0x0002	Match the font type.
IG_PDF_SYSFONT_MATCH_WRITING_MODE	0x0004	Match the writing mode, that is, horizontal or vertical.

1.3.3.6.52 enumIGPDFSysFontPackageType

Flags for packageType of AT_PDE_FONTATTRS.

Values:

IG_PDF_SYSFONT_UNKNOWN	0	Unknown.
IG_PDF_SYSFONT_TYPE1	1	Type1.
IG_PDF_SYSFONT_TRUETYPE	2	TrueType.
IG_PDF_SYSFONT_CID	3	CID.
IG_PDF_SYSFONT_ATC	4	ATC.
IG_PDF_SYSFONT_OCF	5	OCF.
IG_PDF_SYSFONT_OPENTYPE_CFF	6	OpenType CFF.
IG_PDF_SYSFONT_OPENTYPE_CID	7	OpenType CID.
IG_PDF_SYSFONT_OPENTYPE_TT	8	OpenType TT.

1.3.3.6.53 enumIGPDFWordFinderVersion

WordFinder algorithm version.

Values:

IG_PDF_WF_LATEST_VERSION	0	The latest available version.
IG_PDF_WF_VERSION_2	2	Version used for Acrobat 3.x, 4.x.
IG_PDF_WF_VERSION_3	3	Available in Acrobat 5.0 without Accessibility enabled. Includes some improved word piecing algorithms.
IG_PDF_WF_VERSION_4	4	For Acrobat 5.0 with Accessibility enabled. Includes advanced word ordering algorithms in addition to improved word piecing algorithms.

1.3.3.6.54 enumIGPDFWordFlags

Context flags.

Values:

IG_PDF_ORDER	0x2	Use PDF order for text enumeration.
IG_PDF_XY_SORT	0x4	Use XY order for text enumeration.