NEWTONE

イメージ処理コンポーネント

ImageKit10 VCL

コントロールリファレンス

目次

1. コントロールリファレンス

ImageKit10 VCL を使用してのアプリケーションの開発は、必要なコントロールをデザイン時にフォームに配置し、そのコントロールのプロパティの設定やメソッドの実行により各機能を実現します。

【表記上の注意】

●カスタムプロパティ、カスタムメソッドの【書式】中のコントロール名の表記は以下の通りです。

表記	意味
imagekitcontrolname	イメージキットコントロール名を示します。
thumbcontrolname	サムネイルコントロール名を示します。
playcontrolname	動画再生用のプレイコントロール名を示します。
previewcontrolname	Web カメラ用のプレビューコントロール名を示します。
recordcontrolname	Web カメラ用のレコードコントロール名を示します。

※ImageKit7 VCL からサムネイルコントロールは専用パッケージとなりました。

●旧バージョンからの移行について

ImageKit8(VCL 含む)から ImageKit6 互換コントロールとスライドショウコントロールは提供しておりませんので、ImageKit5 や ImageKit6(VCL 含む)から移行される場合、または ImageKit7 ActiveX/VCL で ImageKit6 互換コントロールを使用されていた場合はイメージキットコントロールをご利用ください。スライドショウコントロールを使用していた部分については旧バージョンをそのままご利用ください。なお、同一プロジェクトで旧バージョン(VCL)と本バージョン(VCL)を同時に使用することはできません。

●ActiveX バージョンからの移行について

対象製品:ImageKit5,ImageKit6,ImageKit7 ActiveX,ImageKit8 ActiveX,ImageKit9 ActiveX,ImageKit10 ActiveX

同じメソッドでも引数が追加されたり、新たに戻り値が追加されたりといくつかの変更が施されているものがあります。変更があるものについては【ImageKit? ActiveX との違い】という見出しで始まる簡単な説明がありますので、ご覧ください。

プロパティやメソッドとイベント引数について

文字列型

ActiveX の WideString 型や BSTR 型は、VCL では UnicodeString 型に変更されました。

整数型

ActiveX を C++Builder で使用する際の long 型が int 型に変更されました。

論理型

ActiveX を Delphi で使用する際の WordBool 型が Boolean 型に変更されました。

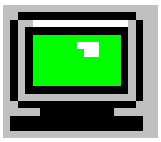
●VCL バージョンからの移行について

対象製品:ImageKit6 VCL,ImageKit7 VCL,ImageKit8 VCL,ImageKit9 VCL

同じプロパティでも型が変更されたり、同じイベントでも引数が追加されたりといくつかの変更が施されているものがあります。変更があるものについては【ImageKit? VCL との違い】という見出しで始まる簡単な説明がありますので、ご覧ください。

文章中に IK5,IK6,IK7,IK8,IK9 という表現がありますが、それぞれ ImageKit5・ImageKit6・ImageKit7・ImageKit8・ImageKit9 の省略形を意味します。

1－1. イメージキットコントロール



イメージキットコントロール

イメージのスキャニング・表示・編集・エフェクト・ファイル処理・描画・印刷の各処理を統合したコントロールです。

継承
TCustomControl
ユニット
ImageKit

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Appearance	コントロールの縁の設定
AutoDisplay	自動でイメージを表示するかどうかの設定
AutoSize	コントロールの大きさを画像サイズに合わせる
BackgroundImageFile	コントロールの背景に表示するイメージのファイル名
BackgroundImageVectorHeight	背景に表示するｲﾒｰｼﾞﾌｧｲﾙがﾍﾞｸﾄﾙ形式の場合の読み込み時の高さ(ﾋﾟｸｾﾙ)
BackgroundImageVectorWidth	背景に表示するｲﾒｰｼﾞﾌｧｲﾙがﾍﾞｸﾄﾙ形式の場合の読み込み時の幅(ﾋﾟｸｾﾙ)
BitCount	基本イメージの１ピクセルあたりのビット数
BlinkSpeed	矩形範囲選択時の選択領域の点滅速度
BorderColor	コントロールの縁の色
BorderVisible	コントロールの縁の描画設定
Canvas.Handle	コントロールのデバイスコンテキスト
Color	コントロールの背景色
Cursor	コントロールの描画領域内におけるマウスカーソルの形状
DispCenterX	コントロールの中心に表示したいイメージの横方向の位置
DispCenterY	コントロールの中心に表示したいイメージの縦方向の位置
DisplayMode	表示モード
DispScaleX	表示のスケーリングの横方向の比率
DispScaleY	表示のスケーリングの縦方向の比率
DispStartX	イメージの横方向の表示開始位置を設定
DispStartY	イメージの縦方向の表示開始位置を設定
DrawVectorColor	ベクトルイメージの表示色
DXFBlack	DXF イメージの白を黒に置き換えて描画するかどうかの設定
Enabled	マウスやキーボードイベントへの応答可否
ErrorStatus	エラーの種類を示す
Grad	目盛の表示単位の設定
GradBackColor	目盛の背景色
GradColor	目盛の色
Gray	グレースケールの有無
Grid	グリッドの表示単位の設定
GridColor	グリッドの色
GridSpace	グリッドの間隔
Handle	コントロールのウィンドウハンドル
ImageAtLeftTop	コントロールのサイズより表示する画像のサイズが小さい場合の表示位置
ImageHandle	基本イメージのメモリハンドルを示す
ImageHdc	表示イメージのメモリデバイスコンテキスト
ImageHeight	基本イメージの縦のピクセル数
ImageKind	基本イメージの種類
ImageSize	基本イメージサイズのバイト数
ImageWidth	基本イメージの横のピクセル数
ImageWidthByte	基本イメージの１ラインのバイト数
InvalidHatchPattern	無効領域をハッチパターンで描画するかどうかの設定

InvalidHatchColor	ハッチパターンの色
LayerNo	処理対象となるイメージ番号(基本イメージ or 階層イメージ)
Mask1632	基本イメージのカラーマスクの種類
MouseDragImage	マウスやタッチ操作でイメージのスクロールを可能にするかどうかの設定
MouseWheel	マウスホイールのスクロールの状態
PalBlue	マウスダウンした点の青のカラー値を示す
PalCount	基本イメージの使用パレット数
PalGreen	マウスダウンした点の緑のカラー値を示す
PalRed	マウスダウンした点の赤のカラー値を示す
Picture	TPicture と ImageKit 間でラスタイメージやベクトルイメージのやり取りを行う
RectBottom	マウスで矩形範囲を指定する時の位置(下/ﾋﾟｸｾﾙ)
RectColor1	矩形選択領域を表す点線の最初の色
RectColor2	矩形選択領域を表す点線の 2 番目の色
RectCursor	マウスドラッグ時におけるマウスカーソルの形状
RectDraw	マウスで矩形範囲の指定を有効にするかどうかの設定
RectDrawFix	マウスで矩形範囲を指定する時のサイズ変更の有無
RectDrawRatio	マウスで矩形範囲を指定する時の縦横比
RectLeft	マウスで矩形範囲を指定する時の位置(左/ﾋﾟｸｾﾙ)
RectRight	マウスで矩形範囲を指定する時の位置(右/ﾋﾟｸｾﾙ)
RectTop	マウスで矩形範囲を指定する時の位置(上/ﾋﾟｸｾﾙ)
Refine1BitImage	1 ビットイメージを縮小表示する際、高精彩に表示するかどうかの設定
RefineColorImage	24,32 ビットカラーイメージを縮小表示する際、高精彩に表示するかどうかの設定
ScrollBar	スクロールバーの表示の設定
ScrollHeight	イメージスクロールの縦ピクセル数の設定
ScrollWidth	イメージスクロールの横ピクセル数の設定
ShowBackgroundImage	コントロールの背景に表示するイメージの表示/非表示の設定
ShowInDisp	基本イメージをイメージキットコントロールに表示するかどうかの設定
ShowInMagnifier	基本イメージを虫眼鏡ウィンドウに表示するかどうかの設定
ShowInPanWindow	基本イメージをパンウィンドウに表示するかどうかの設定
SimpleDispCtrl	基本イメージに限定した表示機能
StretchMode	高精彩表示のモード
ToolTip	マウスを移動させた時の座標値の表示単位の設定
ToolTipColor	マウスを移動させた時の座標値の色
ToolTipXfromMouse	マウス位置からマウス座標値を表示する位置までの X 方向のオフセット
ToolTipYfromMouse	マウス位置からマウス座標値を表示する位置までの Y 方向のオフセット
ToolTipTimeSecond	マウス座標値を表示する秒数
Touch	ピンチイン、ピンチアウトによる拡大・縮小の適用可否
TransBlue	基本イメージ(ラスタ)の透過色の青
TransGreen	基本イメージ(ラスタ)の透過色の緑
Transparent	基本イメージ(ラスタ)の透過の設定
TransRed	基本イメージ(ラスタ)の透過色の赤
VectorAntiAliasDisplay	ベクトルイメージのアンチエイリアス表示の設定
Xdpi	基本イメージの横方向の 1 インチあたりのピクセル数
Ydpi	基本イメージの縦方向の 1 インチあたりのピクセル数

【ImageKit7/8/9/10 ActiveX との違い】

削除されたプロパティ: EnableOLEDrag, MouseCursorFile, RectMouseCursorFile

※MouseCursorFile プロパティは Cursor プロパティで代用

※RectMouseCursorFile プロパティは RectCursor プロパティで代用

変更されたプロパティ:

BackColor --> Color
Hdc --> Canvas.Handle
Hwnd(ImageKit7/8 ActiveX) --> Handle
HWND(ImageKit9/10 ActiveX) --> Handle
MouseCursorType --> Cursor
RectMouseCursorType --> RectCursor

【ImageKit9/10 ActiveX との違い】

変更されたプロパティ:
EnableTouch --> Touch

●メソッド一覧(アルファベット順)

カスタムメソッド	内容
BitmapFromDib	DIB から DDB に変換
BitBlt	デバイスコンテキストのイメージを別のデバイスコンテキストへコピー
ClearClipBrd	クリップボードのクリア
CopyImage	イメージの複製を作成
CreateImage	ラスタイメージを新規に作成
DeleteBitmapObject	ビットマップオブジェクトを削除
DibFromBitmap	DDB から DIB に変換
Display	イメージを描画
DisplayImage	イメージを描画
FreeMemory	メモリハンドルの解放
GetDpi	メモリハンドルから解像度を取得
GetDpiFromHdc	デバイスコンテキストから解像度を取得
GetFromClipBrd	クリップボードからラスタイメージやベクトルイメージの取り込み
GetImageType	メモリハンドルからイメージの情報を取得
GetMemorySize	メモリハンドルから使用しているイメージのメモリサイズを取得
GetOneBitPalCount	1 ビットカラーイメージのパレット 0 とパレット 1 のピクセル数を取得
GetPaletteFromImage	メモリハンドルからパレットデータを取得
GetSystemPalette	現在の PC のシステムパレットを取得
IsClipBrdData	クリップボードのデータチェック
Refresh	イメージの再描画
ScrollImage	実寸表示の時のスクロール
SetDpi	メモリハンドルへ解像度を設定
SetPaletteToImage	メモリハンドルへパレットデータを設定
SetToClipBrd	ラスタイメージやベクトルイメージをクリップボードにコピー
StretchBlt	デバイスコンテキストのイメージを別のデバイスコンテキストへコピー(拡大縮小)
Zoom	指定した範囲の拡大もしくは縮小

【ImageKit7/8/9/10 ActiveX との違い】
変更されたメソッド:
GetPalette --> GetPaletteFromImage
SetPalette --> SetPaletteToImage

●イベント一覧(アルファベット順)

カスタムイベント	内容
AfterScan	スキャンデバイスからイメージを取り込んだ後に発生
BeforeScan	スキャンデバイスからイメージを取り込む前に発生
ClickOnPanWindow	パンウィンドウのマウスクリックイベント
EditToolBar	イメージ編集ツールバーを表示もしくは非表示にすると発生
EndDispImage	イメージを表示した後に発生
EndLoad	ファイル読込処理終了後に発生
EndPopUpMenu	ポップアップメニューの処理が終了した直後に発生
EndRasterToVector	ラスタ化処理の開始前に発生
EndSave	ファイル保存処理終了後に発生
EndVectorToRaster	ベクトル化処理の開始前に発生
GetDragDropFileName	ドラッグ＆ドロップでイメージキットコントロール上にファイルをドロップすると発生
MagnifierMouseDown	虫眼鏡ウィンドウのマウスダウンイベント
MagnifierMouseMove	虫眼鏡ウィンドウのマウスムーブイベント
MagnifierMouseUp	虫眼鏡ウィンドウのマウスアップイベント
MouseDownImage	マウスダウンのイベント
MouseDownSelectLayerImage	階層イメージ上でマウスボタンを押すと発生
MouseMoveImage	マウスムーブのイベント
MouseUpImage	マウスアップのイベント

MouseUpSelectLayerImage	階層イメージ上でマウスボタンを離すと発生
MouseWheelDownImage	マウスホイールを下へ回転すると発生
MouseWheelUpImage	マウスホイールを上へ回転すると発生
PanWindow	パンウィンドウを表示もしくは消去すると発生
Progress	エフェクトおよびファイル処理中に発生
Scanning	スキャンデバイスからイメージ取り込み中に発生
ScrollDispImage	スクロールバーを使用してスクロールすると発生
SelEditFunc	イメージ編集ツールバーのボタンを選択すると発生
SelEditObj	イメージ編集ツールバーのオブジェクトを選択すると発生
StartDispImage	イメージを表示する前に発生
StartLoad	ファイル読込処理開始前に発生
StartPopUpMenu	ポップアップメニューを選択した直後に発生
StartRasterToVector	ラスタ化処理の開始前に発生
StartSave	ファイル保存処理開始前に発生
StartVectorToRaster	ベクトル化処理の開始前に発生

●階層プロパティ一覧(アルファベット順)

カスタムプロパティ	内容
Edit	編集処理で使用
Effect	エフェクト処理(ラスタイメージの加工など)で使用
FileIO	ファイル処理で使用
Layer	イメージの 100 階層用
Magnifier	虫眼鏡処理で使用
PanWindow	パンウィンドウ処理で使用
PrintDraw	図形や文字などの描画およびプリンタの制御に使用
Scan	スキャン処理で使用
Vector	ベクトルイメージの作成およびラスタとベクトルの相互変換

階層下にプロパティ、メソッド(Layer を除く)を持ちます。

【ImageKit7/8/9/10 ActiveX との違い】

変更されたプロパティ:

File --> FileIO

Appearance(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロールの縁の表示を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Appearance** [= *TVIkAppearance*]
(2)Delphi　*imagekitcontrolname*.**Appearance** [= *TVIkAppearance*]

【TVIkAppearance 型】
ユニット
IkInit
type
TVIkAppearance = (vikFlat, vikLowered, vikRaised);

【設定値】

値	説明
vikFlat	Flat
vikLowered	凹
vikRaised	凸

【解説】
BorderVisible プロパティが True の場合に有効です。
vikFlat の場合、**BorderColor** プロパティに設定されている色で縁を描画します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikFlat, ikLowered, ikRaised)。

AutoDisplay(イメージキットコントロール／カスタムプロパティ)

【機能】
自動でイメージを表示するかどうかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**AutoDisplay** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**AutoDisplay** [= *Boolean*]

【設定値】

値	説明
True	自動でイメージを表示する
False	自動でイメージを表示しない

【解説】
デフォルトは True です。
True の場合は、ファイルを読み込むなどして **ImageHandle** プロパティなどが設定されると自動で該当するイメージを表示します。False の場合は、ファイルを読み込むなどして **ImageHandle** プロパティなどが設定されても該当するイメージは表示されません。表示する場合は、**DisplayImage** メソッドを実行する必要があります(表示モードは **DisplayMode** プロパティで設定)。

関連する項目として **ShowInDisp**,**ShowInMagnifier**,**ShowInPanWindow** プロパティも参照してください。

【値の設定】 実行時

【値の参照】 実行時

AutoSize(イメージキットコントロール／カスタムプロパティ)

【機能】
コントロールの大きさを画像サイズに合わせるかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**AutoSize** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**AutoSize** [= *Boolean*]

【設定値】

値	説明
True	コントロールの大きさを画像サイズに合わせる
False	デザイン時のサイズもしくは実行時に設定したサイズ(**Height**,**Width** プロパティ)を維持

【解説】
デフォルトは False です。
True の場合は、画像サイズに合わせてコントロールの右上と右下の位置を変更します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

BackgroundImageFile（イメージキットコントロール／カスタムプロパティ）

【機能】
イメージキットコントロールの背景に表示するイメージファイル名を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**BackgroundImageFile** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**BackgroundImageFile** [= *string*]

【設定値】
コントロールの背景に表示するイメージファイル名。

【解説】
Transparent プロパティが True の場合に有効です。

「背景イメージの規則」
・基本イメージを表示する際にその下部に表示されます。
・サイズが基本イメージよりも大きい場合は、基本イメージのサイズに合わせられます。

コード例：
(1)C++Builder
```
//背景イメージの設定
VImageKit1->BackgroundImageFile = "C:¥¥BackImage.bmp";
//基本イメージの透明色を白色に設定
VImageKit1->TransBlue = 255;
VImageKit1->TransGreen = 255;
VImageKit1->TransRed = 255;
VImageKit1->Transparent = true;
```
(2)Delphi
```
//背景イメージの設定
VImageKit1.BackgroundImageFile := 'C:¥BackImage.bmp';
//基本イメージの透明色を白色に設定
VImageKit1.TransBlue := 255;
VImageKit1.TransGreen := 255;
VImageKit1.TransRed := 255;
VImageKit1.Transparent := True;
```

【値の設定】 実行時

【値の参照】 実行時

BackgroundImageVectorHeight,BackgroundImageVectorWidth(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロールの背景に表示するイメージファイルがベクトル形式の場合、読み込み時の高さと幅を取得または設定します。

【書式】
※**BackgroundImageVectorHeight** にて説明(**BackgroundImageVectorWidth** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**BackgroundImageVectorHeight** [= *int*]
(2)Delphi　*imagekitcontrolname*.**BackgroundImageVectorHeight** [= *Integer*]

【設定値】
コントロールの背景に表示するベクトルイメージの縦横のピクセル数。
BackgroundImageVectorHeight はイメージの高さ(ピクセル)、デフォルトは 0。
BackgroundImageVectorWidth はイメージの幅(ピクセル)、デフォルトは 0。

【解説】
BackgroundImageFileプロパティに設定されたファイルがDXF,SXFの場合
BackgroundImageVectorHeight もしくは **BackgroundImageVectorWidth** が 0 以下の場合は高さ 600 幅 800 の値を使用します。
BackgroundImageFileプロパティに設定されたファイルがEMF,SVG,WMFの場合
BackgroundImageVectorHeight もしくは **BackgroundImageVectorWidth** が 0 以下の場合はファイルに格納されているサイズを使用します。
(注意)
読み込まれるイメージの縦横のサイズは、設定した矩形内(BackgroundImageVectorWidth * BackgroundImageVectorHeight)で縦横比を考慮した最も大きなサイズとなりますので、設定したサイズ通りにならない場合があります。

ベクトルイメージを背景とする場合は **BackgroundImageVectorHeight** プロパティと **BackgroundImageVectorWidth** プロパティを **BackgroundImageFile** プロパティよりも先に設定する必要があります。

コード例:
(1)C++Builder
```
//背景イメージの設定
VImageKit1->BackgroundImageVectorWidth = 640;
VImageKit1->BackgroundImageVectorHeight = 480;
VImageKit1->BackgroundImageFile = "C:¥¥BackImage.emf";
//基本イメージの透明色を白色に設定
VImageKit1->TransBlue = 255;
VImageKit1->TransGreen = 255;
VImageKit1->TransRed = 255;
VImageKit1->Transparent = true;
```
(2)Delphi
```
//背景イメージの設定
VImageKit1.BackgroundImageVectorWidth := 640;
VImageKit1.BackgroundImageVectorHeight := 480;
VImageKit1.BackgroundImageFile := 'C:¥BackImage.emf';
//基本イメージの透明色を白色に設定
VImageKit1.TransBlue := 255;
VImageKit1.TransGreen := 255;
VImageKit1.TransRed := 255;
VImageKit1.Transparent := True;
```

【値の設定】 実行時

【値の参照】 実行時

BitCount(イメージキットコントロール／カスタムプロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの 1 ピクセルあたりのビット数を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**BitCount** [= *short*]
(2)Delphi　*imagekitcontrolname*.**BitCount** [= *Smallint*]

【参照値】
イメージの 1 ピクセルあたりのビット数。

値	説明
1	2 値
4	16 色
8	256 色
16	16 ビットカラー
24	24 ビットカラー
32	32 ビットカラー

【解説】
ImageHandle プロパティに値が設定されると、BitCount プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

BlinkSpeed(イメージキットコントロール／カスタムプロパティ)

【機能】
矩形範囲選択時の選択領域の点滅速度を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**BlinkSpeed** [= *TVIkBlinkSpeed*]
(2)Delphi　*imagekitcontrolname*.**BlinkSpeed** [= *TVIkBlinkSpeed*]

【TVIkBlinkSpeed 型】
ユニット
IkInit
type
TVIkBlinkSpeed = (vikSlow, vikMedium, vikFast);

【設定値】

値	説明
vikSlow	低速
vikMedium	中速
vikFast	高速

【解説】
RectDraw プロパティが True で矩形領域が選択されている場合に有効です。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikSlow, ikMedium, ikFast)。

BorderColor,Color(イメージキットコントロール／カスタムプロパティ)

【機能】
色情報を取得または設定します。

【書式】
※**BorderColor** にて説明(**Color** も同様な使い方)
(1)C++Builder　*displaycontrolname*->**BorderColor** [= *TColor*]
(2)Delphi　*displaycontrolname*.**BorderColor** [= *TColor*]

【設定値】
BorderColor はイメージキットコントロールの縁の色。
Color はイメージキットコントロールの背景色。

【解説】
値を設定する場合は、色定数(clRed など)や **RGB**(Red,Green,Blue)の戻り値などを与えます。
BorderColor プロパティは **Appearance** プロパティが vikFlat で **BorderVisible** プロパティが True の場合に有効です。
TColor 型については Delphi や C++Builder のヘルプを参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
Color プロパティの名称が **BackColor** から変更されました。

BorderVisible（イメージキットコントロール／カスタムプロパティ）

【機能】
イメージキットコントロールの縁を描画するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**BorderVisible** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**BorderVisible** [= *Boolean*]

【設定値】

値	説明
True	縁を描画する
False	縁を描画しない

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Canvas.Handle(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロールのデバイスコンテキストを取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Canvas**->**Handle** [*= HDC*]
(2)Delphi　*imagekitcontrolname*.**Canvas**.**Handle** [*= HDC*]

【参照値】
イメージキットコントロールのデバイスコンテキスト。

【解説】
Canvas.**Handle** プロパティはイメージキットコントロールのデバイスコンテキストを示すので、**Canvas** プロパティのメソッドや **PrintDraw** プロパティに実装されているメソッドを使用してイメージキットコントロール上に描画することができます。
Canvas については Delphi や C++Builder のヘルプの TCanvas を参照してください。

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
プロパティの名称が **Hdc** から変更されました。

Cursor(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロールの描画領域内におけるマウスカーソルの形状を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Cursor** [= *TCursor*]
(2)Delphi　*imagekitcontrolname*.**Cursor** [= *TCursor*]

【設定値】
crDefault や crArrow など。詳しくは Delphi や C++Builder のヘルプを参照のこと。

【解説】
イメージ編集ツールバーのそれぞれのボタンに割り当てられているカーソル(ImageKit10 のコントロールで定義されているカーソル)を有効にする場合は、crDefault を設定してください。
カスタムカーソルを割り当てる場合は、Delphi や C++Builder のヘルプの TControl.Cursor の例を参照してください。
また、このプロパティはマウスドラッグ時のマウスカーソル形状を表す **RectCursor** プロパティとは異なります。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
MouseCursorFile,MouseCursorType プロパティの機能が **Cursor** プロパティに統一されました。

DispCenterX,DispCenterY（イメージキットコントロール／カスタムプロパティ）

【機能】
コントロールの中心に表示したいイメージの縦・横の位置（ピクセル）を取得または設定します。

【書式】
※DispCenterX にて説明（DispCenterY も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->DispCenterX [= *int*]
(2)Delphi　*imagekitcontrolname*.DispCenterX [= *Integer*]

【設定値】
コントロールの中心に表示したい縦横の位置（0 以上）をピクセルで設定します。

【解説】
設定された DispCenterX,DispCenterY プロパティの位置をイメージキットコントロールの中心に合わせてイメージを表示します（表示するには DisplayImage メソッドを実行する必要があります）。
実寸表示の時に DispCenterX,DispCenterY プロパティは有効になります。
ただし、DispStartX,DispStartY プロパティに 0 以外が設定されている場合には、DispCenterX,DispCenterY プロパティは無効になります。
また、DispCenterX、DispCenterY プロパティを設定してイメージを表示した後で、イメージをスクロールすると DispStartX,DispStartY プロパティが設定されるため、再度 DispCenterX,DispCenterY プロパティを有効にして表示する場合は、DispStartX,DispStartY プロパティを 0 にし DispCenterX,DispCenterY プロパティを設定する必要があります。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

DisplayMode(イメージキットコントロール／カスタムプロパティ)

【機能】
表示モードを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**DisplayMode** [= *TVIkDispMode*]
(2)Delphi　*imagekitcontrolname*.**DisplayMode** [= *TVIkDispMode*]

【TVIkDispMode 型】
ユニット
IkInit
type
TVIkDispMode = (vikClear, vikScale, vikStretch, vikActualSize, vikFitToWidth, vikFitToHeight);

【設定値】

値	説明
vikClear	クリア
vikScale	スケール(イメージのサイズがコントロールのサイズより大きい場合は縮小して表示)
vikStretch	ストレッチ(コントロールのサイズに合わせてイメージをはめこむ)
vikActualSize	実寸(**DispScaleX**,**DispScaleY** プロパティによる)
vikFitToWidth	横幅に合わせる(コントロールの **Width** プロパティに合わせてイメージをはめこむ)
vikFitToHeight	縦幅に合わせる(コントロールの **Height** プロパティに合わせてイメージをはめこむ)

【解説】
DisplayImage メソッドを実行する際の表示モードになります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikClear, ikScale, ikStretch, ikActualSize, ikFitToWidth, ikFitToHeight)。

DispScaleX,DispScaleY（イメージキットコントロール／カスタムプロパティ）

【機能】
イメージキットコントロールに表示するイメージの縦横のスケーリングの比率を取得または設定します。

【書式】
※**DispScaleX** にて説明（**DispScaleY** も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->**DispScaleX** [*= double*]
(2)Delphi　*imagekitcontrolname*.**DispScaleX** [*= Double*]

【設定値】
DispScaleX は横方向のスケーリングの比率（実寸は 1.0）。
DispScaleY は縦方向のスケーリングの比率（実寸は 1.0）。

【解説】
実寸表示の時に有効になります。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

DispStartX,DispStartY(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロール上に表示するイメージの縦・横の開始位置(ピクセル)を取得または設定します。

【書式】
※DispStartX にて説明(DispStartY も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->DispStartX [= *int*]
(2)Delphi　*imagekitcontrolname*.DispStartX [= *Integer*]

【設定値】
イメージの縦横の表示開始位置(0 以上)をピクセルで設定します。

【解説】
DispStartX,DispStartY プロパティを設定することにより、大きなイメージでも特定の部分から表示を開始することができます。
実寸表示の場合のみ、DispStartX,DispStartY プロパティは有効になります。
ただし、DispCenterX,DispCenterY プロパティに 0 以外が設定されている場合には、DispStartX,DispStartY プロパティは無効になります。
DispStartX,DispStartY プロパティは以前の ImageKit で提供していたディスプレイコントロールの StartX,StartY プロパティに相当します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

DrawVectorColor(イメージキットコントロール／カスタムプロパティ)

【機能】
ベクトルイメージを描画する色を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**DrawVectorColor** [= *TColor*]
(2)Delphi　*imagekitcontrolname*.**DrawVectorColor** [= *TColor*]

【設定値】
描画色。デフォルトは-1 です。
デフォルトの場合、イメージの持つ描画色を使用します。

【解説】
値を設定する場合は、-1 や色定数(clRed など)および **RGB**(Red,Green,Blue)の戻り値などを与えます。
-1 以外の値を設定すると単一色で描画されます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

DXFBlack（イメージキットコントロール／カスタムプロパティ）

【機能】
DXF イメージを描画する際に白を黒に置き換えて描画するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**DXFBlack** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**DXFBlack** [= *Boolean*]

【設定値】

値	説明
True	置き換えを有効
False	置き換えを無効

【解説】
デフォルトは True です。
True の場合は、DXF イメージの白を黒に置き換えて描画します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Enabled(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントールのマウスやキーボードイベントに応答するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Enabled** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Enabled** [= *Boolean*]

【設定値】

値	説明
True	マウスやキーボードイベントを有効
False	マウスやキーボードイベントを無効

【解説】
True を設定するとマウスダウンやキーダウンなどのイベントに応答できますが、False の場合は該当イベントが発生しません。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ErrorStatus(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロールを使用して起きたエラーの種類を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ErrorStatus** [= *short*]
(2)Delphi　*imagekitcontrolname*.**ErrorStatus** [= *Smallint*]

【参照値】

値	説明
0	正常終了(エラーなし)
1	キャンセル
2	その他のエラー
3	メモリエラー
4	サポート外機能
5	設定値が不正
6	ファイルが不正
7	イメージが不正
8	Twain_32.dll または TWAINDSM.dll がロードできない
9	描画エラー
10	対象外イメージ
11	実行順序エラー
12	ADF に用紙がセットされていない(UI 非表示のみ)
13	デジタルカメラに画像がない(UI 非表示のみ)
14	DLL がロードされていない
15	通信エラー
16	ファイルが見つからない
17	ファイルが壊れている
18	タイムオーバーで処理をキャンセル
19	パスワード、ユーザ名が異なる(アクセスができない)
20	ファイルがオープンできない
21	ファイルが作成できない
22	ファイルが書き込みできない
23	ファイルが読み込みできない
24	表示イメージデータがない
101	Twain_32.dll または TWAINDSM.dll にエントリポイントがない
102	データソースマネージャやデータソースがオープンされていない、もしくはオープンできない
103	データソースがない
104	データソースは既に最大数のアプリケーションに接続されている
105	データソースまたはデータソースマネージャでエラーが発生
106	認識できないトリプレット(TWAIN - DG_XXX/DAT_XXX/MSG_XXX)
107	スキャンデバイスの稼動状態を確認
108	フィーダーに用紙が詰まった
109	フィーダーに複数の用紙が取り込まれた
110	解像度が不正
111	カバーが開いている
112	原稿の角が傷んでいる
113	原稿をキャプチャー中のフォーカスエラー
114	原稿がとても明るい
115	原稿がとても暗い
116	イメージ情報が取得できないため転送不可

【解説】
イメージキットコントロールのメソッドを実行することにより、値を参照できます。
13 の場合は、デジタルカメラの TWAIN ドライバが TWAIN 規約のバージョン 1.7 以降に対応している場合が対象で、108・

109 は TWAIN ドライバが TWAIN 規約のバージョン 1.8 以降に対応している場合が対象となります。
110 は EPSON 製スキャナ用ドライバご利用時にモアレ除去機能を使用した場合が対象です。
111～115 は TWAIN ドライバが TWAIN 規約のバージョン 2.0 以降に対応している場合が対象で、116 は TWAIN ドライバが TWAIN 規約のバージョン 2.1 以降に対応している場合が対象となります。

【値の設定】 不可

【値の参照】 実行時

Grad,Grid,ToolTip(イメージキットコントロール／カスタムプロパティ)

【機能】
目盛、グリッド、マウスの座標値の表示単位を取得または設定します。

【書式】
※**Grad** にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Grad** [= *TVIkUnit*]
(2)Delphi　*imagekitcontrolname*.**Grad** [= *TVIkUnit*]

【TVIkUnit 型】
ユニット
IkInit
type
TVIkUnit = (vikNone, vikPixel, vikmm, vikInch);

【設定値】

値	説明
vikNone	なし
vikPixel	ピクセル
vikmm	ミリメートル
vikInch	インチ

【解説】
Grad プロパティは目盛(イメージキットコントロールの左と上に付加)、**Grid** プロパティはグリッド(描画領域に格子の線を描画)、**ToolTip** プロパティはマウスを移動させた時の座標値を表します。
値が vikNone の場合は何も表示しませんが、それ以外の場合は目盛(グリッドもしくはマウス座標値)を表示する単位を表します。

(注意)
Grad,**Grid** プロパティについては vikmm(ミリメートル)を設定しても表示はセンチメートル単位となります。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikNone, ikPixel, ikmm, ikInch)。

GradBackColor,GradColor,GridColor,InvalidHatchColor,RectColor1,RectColor2,ToolTipColor (イメージキットコントロール／カスタムプロパティ)

【機能】
色情報を取得または設定します。

【書式】
※GradBackColor にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->GradBackColor [= *TColor*]
(2)Delphi　*imagekitcontrolname*.GradBackColor [= *TColor*]

【設定値】
GradBackColor は目盛の背景色。
GradColor は目盛の色。
GridColor はグリッドの色。
InvalidHatchColor はハッチパターンの色。
RectColor1 は矩形選択領域を表す点線の最初の色。
RectColor2 は矩形選択領域を表す点線の 2 番目の色。
ToolTipColor はマウス移動時の座標値を表示する色。

【解説】
値を設定する場合は、色定数(clRed など)や RGB(Red,Green,Blue)の戻り値などを与えます。

GradBackColor,GradColor プロパティは Grad プロパティが vikNone 以外の場合に有効です。
GridColor プロパティは Grid プロパティが vikNone 以外の場合に有効です。
InvalidHatchColor プロパティは InvalidHatchPattern プロパティが vikNonHatch 以外の場合に有効です。
RectColor1,RectColor2 プロパティは RectDraw プロパティが True の場合に有効です。
ToolTipColor プロパティは ToolTip プロパティが vikNone 以外の場合に有効です。

TColor 型については Delphi や C++Builder のヘルプを参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Gray(イメージキットコントロール／カスタムプロパティ)

【機能】
ImageHandle プロパティに設定されたイメージがグレースケールであるかどうかを取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Gray** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Gray** [= *Boolean*]

【参照値】

値	説明
True	グレースケール
False	グレースケールでない

【解説】
ImageHandle プロパティに値が設定されると、Gray プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

GridSpace(イメージキットコントロール／カスタムプロパティ)

【機能】
グリッドの間隔を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**GridSpace** [= *double*]
(2)Delphi　*imagekitcontrolname*.**GridSpace** [= *Double*]

【設定値】
Grid プロパティで設定された単位に基づく値。

【解説】
Grid プロパティが vikPixel の場合はピクセル単位となります。
小さい値を設定した場合や実寸表示の縮小率によっては自動的に調整されます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Handle(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロールのウィンドウハンドルを取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Handle** [= *HWND*]
(2)Delphi　*imagekitcontrolname*.**Handle** [= *HWND*]

【参照値】
イメージキットコントロールのウィンドウハンドル。

【解説】
Handle プロパティはイメージキットコントロールのウィンドウハンドルを示すので、この値からウィンドウハンドルを引数にとる、各種 WindowsAPI を呼び出すことができます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8 ActiveX との違い】
プロパティの名称が **Hwnd** から変更されました。

【ImageKit9/10 ActiveX との違い】
プロパティの名称が **HWND** から変更されました。

ImageAtLeftTop(イメージキットコントロール／カスタムプロパティ)

【機能】
コントロールのサイズより画像のサイズが小さい場合の表示位置を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ImageAtLeftTop** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**ImageAtLeftTop** [= *Boolean*]

【設定値】

値	説明
True	コントロールの左上から画像を表示する
False	コントロールの中央に画像を表示する

【解説】
デフォルトは False です。
コントロールのサイズより画像のサイズが小さく、スケール表示および実寸表示時に有効です。
表示モードについては **DisplayMode** プロパティをご覧ください。
コントロールのサイズより画像のサイズが大きい場合は無効です。その場合の表示位置は、**DispCenterX**,**DispCenterY**, **DispStartX**,**DispStartY** プロパティが有効になります。

【値の設定】 実行時

【値の参照】 不可

ImageHandle(イメージキットコントロール／カスタムプロパティ)

【機能】
基本イメージのメモリハンドルを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ImageHandle** [= *NativeUInt*]
(2)Delphi　*imagekitcontrolname*.**ImageHandle** [= *THandle*]

【設定値】
基本となるイメージのメモリハンドル。

【解説】
ImageHandle プロパティにメモリハンドルを直接設定することも可能ですが、**LayerNo** プロパティに-1 を設定してファイルをロードすると、自動的に値が設定されます。
以前の ImageKit で提供していたディスプレイコントロールでは基本イメージはラスタのみでしたが、イメージキットコントロールではラスタとベクトルの区別なく設定が可能です。
編集モードにして **ImageHdc** プロパティを操作する場合は、**ImageHandle** プロパティの指すメモリハンドルを操作しても画面には反映されません。
ImageHandle プロパティの指すイメージを無効にする場合は 0 を設定してください。そうすることにより自動的にメモリが解放されます。また、**ImageHandle** プロパティについては **FreeMemory** メソッドを使用して明示的にメモリを解放しなくても、再設定時やコントロールが破棄される時に自動的に解放されます。

【値の設定】 実行時

【値の参照】 実行時

ImageHdc(イメージキットコントロール／カスタムプロパティ)

【機能】
表示イメージのメモリデバイスコンテキストを取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ImageHdc** [= *HDC*]
(2)Delphi　*imagekitcontrolname*.**ImageHdc** [= *HDC*]

【参照値】
表示イメージのメモリデバイスコンテキスト。

【解説】
ImageHdc プロパティは **ImageHandle** プロパティにラスタイメージが設定され、かつ **Edit.EditEnable** プロパティが True の場合に使用可能となります。ただし、その分余計にメモリを消費してしまいますので注意してください。
また、表示イメージのメモリデバイスコンテキストを示すので、**PrintDraw** プロパティに実装されているメソッドを使用してイメージ上に描画することができます。
ImageHdc プロパティは以前の ImageKit で提供していたディスプレイコントロールの **HdcMemory** プロパティに相当します。

【値の設定】 不可

【値の参照】 実行時

ImageHeight,ImageWidth（イメージキットコントロール／カスタムプロパティ）

【機能】
ImageHandle プロパティに設定されたイメージの縦横のサイズをピクセル単位で取得します。

【書式】
※ImageHeight にて説明（ImageWidth も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->**ImageHeight** [*= int*]
(2)Delphi　*imagekitcontrolname*.**ImageHeight** [*= Integer*]

【参照値】
ImageHeight は縦方向のピクセル数。
ImageWidth は横方向のピクセル数。

【解説】
ImageHandle プロパティに値が設定されると、ImageHeight,ImageWidth プロパティは自動的に更新されます。

【値の設定】 不可

【値の参照】 実行時

ImageKind(イメージキットコントロール／カスタムプロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの種類を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ImageKind** [= *TVIkImageKind*]
(2)Delphi　*imagekitcontrolname*.**ImageKind** [= *TVIkImageKind*]

【TVIkImageKind 型】
ユニット
IkInit
type
TVIkImageKind = (vikInvalidImage, vikRasterImage, vikVectorWMF, vikVectorEMF, vikVectorDXF, vikVectorSVG, vikVectorSXF);

【参照値】

値	説明
vikInvalidImage	不明
vikRasterImage	ラスタイメージ
vikVectorWMF	WMF
vikVectorEMF	EMF
vikVectorDXF	DXF
vikVectorSVG	SVG
vikVectorSXF	SXF

【解説】
ImageHandle プロパティに値が設定されると、**ImageKind** プロパティは自動的に更新されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikErrorImage, ikRasterImage, ikVectorWMF, ikVectorEMF, ikVectorDXF, ikVectorSVG, ikVectorSXF)。
・ikErrorImage が vikInvalidImage に変更されました。

ImageSize（イメージキットコントロール／カスタムプロパティ）

【機能】
ImageHandle プロパティに設定されたイメージのサイズをバイト単位で取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ImageSize** [*= int*]
(2)Delphi　*imagekitcontrolname*.**ImageSize** [*= Integer*]

【参照値】
イメージサイズのバイト数。

【解説】
ImageHandle プロパティに値が設定されると、ImageSize プロパティは自動的に更新されます。

【値の設定】 不可

【値の参照】 実行時

ImageWidthByte(イメージキットコントロール／カスタムプロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの 1 ラインのバイト数を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ImageWidthByte** [= *int*]
(2)Delphi　*imagekitcontrolname*.**ImageWidthByte** [= *Integer*]

【参照値】
イメージの 1 ラインのバイト数。

【解説】
ImageHandle プロパティに値が設定されると、ImageWidthByte プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

InvalidHatchPattern(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロールの無効領域をハッチパターンで描画するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**InvalidHatchPattern** [= *TVIkInvalidHatch*]
(2)Delphi　*imagekitcontrolname*.**InvalidHatchPattern** [= *TVIkInvalidHatch*]

【TVIkInvalidHatch 型】
ユニット
ImageKit
type
TVIkInvalidHatch = (vikNonHatch, vikShowHatch, vikTransparentHatch);

【設定値】

値	説明
vikNonHatch	無効領域をハッチパターンで描画しない(**BackColor** プロパティで塗り潰す)
vikShowHatch	無効領域をハッチパターンで描画する
vikTransparentHatch	無効領域を透明にする(表示モードがスケール時に有効)

【解説】
デフォルトは vikShowHatch です。
vikShowHatch の場合、**InvalidHatchColor** プロパティに設定した色で無効領域を描画します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikNonHatch, ikShowHatch, ikTransparentHatch)。

LayerNo(イメージキットコントロール／カスタムプロパティ)

【機能】
処理対象となるイメージ番号を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**LayerNo** [*= short*]
(2)Delphi　*imagekitcontrolname*.**LayerNo** [*= Smallint*]

【設定値】
イメージ番号(デフォルトは-1)。

-1:基本イメージ
0～99:階層イメージ

【解説】
ファイルの読み込みや保存処理、もしくはエフェクト処理などで対象となるイメージ番号を示します。
例えば、**LayerNo** プロパティに-1 を設定してファイルをロードすると **ImageHandle** プロパティに値が設定され、**LayerNo** プロパティに 0～99 を設定してファイルをロードすると **Layer[LayerNo].ImageHandle** プロパティに値が設定されます。

【値の設定】 実行時

【値の参照】 実行時

Mask1632(イメージキットコントロール／カスタムプロパティ)

【機能】
ImageHandle プロパティに設定された 16 ビットカラーおよび 32 ビットカラーのイメージのカラーマスクの種類を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Mask1632** [*= short*]
(2)Delphi　*imagekitcontrolname*.**Mask1632** [*= Smallint*]

【参照値】
16,32 ビットカラーの時のカラーマスクの種類。

値	説明
0	BitCount プロパティが 16,32 でない
1	カラーマスクあり(RGB555 フォーマット)
2	カラーマスクあり(RGB565 フォーマット)
3	カラーマスクなし(RGB555 フォーマット)
4	カラーマスクあり(RGB888 フォーマット)
5	カラーマスクなし(RGB888 フォーマット)

【解説】
ImageHandle プロパティに値が設定されると、Mask1632 プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

MouseDragImage（イメージキットコントロール／カスタムプロパティ）

【機能】
マウスやタッチ操作でイメージのスクロールを可能にするかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**MouseDragImage** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**MouseDragImage** [= *Boolean*]

【設定値】

値	説明
True	スクロール可
False	スクロール不可

【解説】
True の場合、イメージキットコントロールのイメージをマウスやタッチ操作でドラッグすることによりスクロールが可能です。
MouseDragImage プロパティは以前の ImageKit で提供していたディスプレイコントロールの **ScrollDrag** プロパティに相当します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

MouseWheel(イメージキットコントロール／カスタムプロパティ)

【機能】
マウスホイールのスクロールの状態を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**MouseWheel** [= *TVIkMouseWheel*]
(2)Delphi　*imagekitcontrolname*.**MouseWheel** [= *TVIkMouseWheel*]

【TVIkMouseWheel 型】
ユニット
IkInit
type
TVIkMouseWheel = (vikDisable, vikVertical, vikHorizontal);

【設定値】

値	説明
vikDisable	スクロールを無効
vikVertical	垂直方向にスクロール
vikHorizontal	水平方向にスクロール

【解説】
vikDisable 以外を設定するとマウスホイールによるスクロールが可能です。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikDisable, ikVertical, ikHorizontal)。

PalBlue,PalGreen,PalRed（イメージキットコントロール／カスタムプロパティ）

【機能】
イメージキットコントロール上の基本イメージをマウスでダウンした場合の、その位置の赤・緑・青（RGB）の各カラー値を取得します。

【書式】
※PalBlue にて説明（PalGreen、PalRed も同様な使い方）
(1)C++Builder　　*imagekitcontrolname*->PalBlue [= *short*]
(2)Delphi　　*imagekitcontrolname*.PalBlue [= *Smallint*]

【参照値】
0～255。

【解説】
PalBlue,PalGreen,PalRed の各プロパティは、主に「透明 GIF 形式」を利用する場合で、どの色を透明色にするか実際のイメージを見ながらその色を決定する場合に利用します。
「透明 GIF 形式」で透明色を設定する場合は、FileIO.PalBlue,FileIO.PalGreen,FileIO.PalRed の各プロパティに該当する青・緑・赤の各カラー値をセットします。
その際、FileIO.PalBlue,FileIO.PalGreen,FileIO.PalRed の各プロパティに直接各カラー値をセットしても構いませんが、イメージキットコントロールの PalBlue,PalGreen,PalRed の各プロパティの各カラー値をそのままセットすることで、簡単に実際のイメージを見ながら決定した各カラー値を透明色とすることができます。
PalBlue,PalGreen,PalRed の各プロパティ値の参照はイメージキットコントロールのマウスダウンイベントなどで取得するとよいでしょう。
また、イメージキットコントロールの任意のピクセルの RGB を取得するには、PrintDraw.GetPixel メソッドを使用します。
PalBlue,PalGreen,PalRed の各プロパティは、値の参照専用で値を設定しても無効です。

【値の設定】 不可

【値の参照】 実行時

PalCount（イメージキットコントロール／カスタムプロパティ）

【機能】
ImageHandle プロパティに設定されたイメージの使用パレット数を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PalCount** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PalCount** [= *Smallint*]

【参照値】
イメージの使用パレット数。
BitCount プロパティが 8 以下の場合が対象となり、それ以外は 0 となります。

【解説】
ImageHandle プロパティに値が設定されると、PalCount プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

Picture(イメージキットコントロール／カスタムプロパティ)

【機能】
TPicture と ImageKit 間でラスタイメージやベクトルイメージのやり取りをするために使用します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Picture** [*= TPicture**]
(2)Delphi　*imagekitcontrolname*.**Picture** [*= TPicture*]

【設定値】
TPicture。
TPicture については Delphi や C++Builder のヘルプを参照してください。

【解説】
ラスタイメージをやり取りするコードを示します。

C++Builder の例:
(1)TImage → TVImageKit
```
VImageKit1->Picture = Image1->Picture;
```
(2)TVImageKit → TImage
```
Image1->Picture = VimageKit1->Picture;
```

Delphi の例:
(1)TImage → TVImageKit
```
VImageKit1.Picture := Image1.Picture;
```
(2)TVImageKit → TImage
```
Image1.Picture := VImageKit1.Picture;
```

【値の設定】 実行時

【値の参照】 実行時

RectBottom,RectLeft,RectRight,RectTop（イメージキットコントロール／カスタムプロパティ）

【機能】
マウスでイメージの矩形範囲指定時の頂点 4 位置をピクセル単位で示します。

【書式】
※**RectBottom** にて説明（**RectLeft**,**RectRight**,**RectTop** も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->**RectBottom** [= *int*]
(2)Delphi　*imagekitcontrolname*.**RectBottom** [= *Integer*]

【設定値】
イメージキットコントロール上のイメージの左上を原点(0,0)としてピクセル単位で設定します。

【解説】
イメージキットコントロールの **RectDraw** プロパティを True にすることにより、基本イメージの任意の矩形範囲をマウスで指定することができます。
その際の任意の矩形範囲（下、左、右、上）を表すのが **RectBottom**,**RectLeft**,**RectRight**,**RectTop** の各プロパティです。
また、**RectDraw** プロパティを True に設定し、その後マウスをドラッグして離すと、その都度その矩形範囲の頂点 4 位置が **RectBottom**,**RectLeft**,**RectRight**,**RectTop** の各プロパティに格納されます。
なお、矩形選択時の矩形の色は **RectColor1**,**RectColor2** プロパティで指定した色となり、線種は点線となります。点線の点滅速度は **BlinkSpeed** プロパティで設定します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

RectCursor(イメージキットコントロール／カスタムプロパティ)

【機能】
マウスドラッグ時におけるマウスカーソルの形状を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**RectCursor** [= *TCursor*]
(2)Delphi　*imagekitcontrolname*.**RectCursor** [= *TCursor*]

【設定値】
crDefault や crArrow など。詳しくは Delphi や C++Builder のヘルプを参照のこと。

【解説】
デフォルトは crSizeAll です。設定したカーソル形状が有効になるのは、**RectDraw** プロパティが True で矩形領域が選択されていて(**RectLeft,RectTop,RectRight,RectBottom** プロパティが設定されている場合)その内部でマウスのボタンが押された場合やイメージ編集ツールバーを使用して編集している場合、および **MouseDragImage** プロパティが True でマウスドラッグによるスクロール処理を行った場合です。
カスタムカーソルを割り当てる場合は、Delphi や C++Builder のヘルプの TControl.Cursor の例を参照してください。
また、イメージキットコントロールの描画領域内におけるマウスカーソルの形状は、**Cursor** プロパティを設定してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
RectMouseCursorFile,RectMouseCursorType プロパティの機能が **RectCursor** プロパティに統一されました。

RectDraw(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロール上の基本イメージの任意の矩形範囲をマウスで指定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**RectDraw** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**RectDraw** [= *Boolean*]

【設定値】

値	説明
True	マウスでイメージの矩形範囲指定を有効
False	マウスでイメージの矩形範囲指定を無効

【解説】
RectDraw プロパティを True にすることにより、基本イメージの任意の矩形範囲をマウスで指定することができます。
その際の任意の矩形範囲(下、左、右、上)を表すのが **RectBottom**,**RectLeft**,**RectRight**,**RectTop** の各プロパティです。
また、**RectDraw** プロパティを True に設定し、その後マウスをドラッグして離すと、その都度その矩形範囲の頂点 4 位置が **RectBottom**,**RectLeft**,**RectRight**,**RectTop** の各プロパティに格納されます。
なお、矩形選択時の矩形の色は **RectColor1**,**RectColor2** プロパティで指定した色となり、線種は点線となります。点線の点滅速度は **BlinkSpeed** プロパティで設定します。
MouseDragImageプロパティがTrueの場合、**RectDraw**プロパティがTrueでも矩形範囲の選択や移動はできません。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

RectDrawFix(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロール上の基本イメージの任意の矩形範囲をマウスでサイズ変更できるかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**RectDrawFix** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**RectDrawFix** [= *Boolean*]

【設定値】

値	説明
True	イメージの矩形範囲をサイズ変更不可にする
False	イメージの矩形範囲をサイズ変更可にする

【解説】
RectDrawFix プロパティに True を設定する場合は、**RectDraw** プロパティにも True を設定する必要があります。そうすることにより、イメージキットコントロール上のイメージの任意の矩形範囲のサイズを変更不可にすることができます。(矩形範囲の移動は可)
また、当プロパティに値を設定しない場合はサイズ変更可(False)となります。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

RectDrawRatio（イメージキットコントロール／カスタムプロパティ）

【機能】
マウスで矩形範囲を指定する時の縦横比を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**RectDrawRatio** [= *double*]
(2)Delphi　*imagekitcontrolname*.**RectDrawRatio** [= *Double*]

【設定値】
矩形の縦横比（デフォルトは 0）

【解説】
RectDrawRatio プロパティを有効にする場合は、**RectDraw** プロパティを True にする必要があります。
RectDrawRatio プロパティが 0 の場合はマウス操作に依存しますが、それ以外の場合は指定した縦横比で矩形が大きくなったり小さくなったりします。

表示イメージと同じ縦横比で矩形範囲を指定するコード例：
(1)C++Builder

```
bool Ret;

VImageKit1->LayerNo = -1;
Ret = VImageKit1->GetImageType();
if (Ret == false) return;
VImageKit1->RectDrawRatio = VImageKit1->ImageHeight / VImageKit1->ImageWidth;
VImageKit1->RectDraw = true;
```

(2)Delphi

```
Ret: Boolean;

VImageKit1.LayerNo := -1;
Ret := VImageKit1.GetImageType;
if Ret = False then Exit;
VImageKit1.RectDrawRatio := VImageKit1.ImageHeight / VImageKit1.ImageWidth;
VImageKit1.RectDraw := True;
```

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Refine1BitImage(イメージキットコントロール／カスタムプロパティ)

【機能】
1ビットイメージや4,8ビットグレースケールイメージを縮小表示する際、高精彩に表示するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Refine1BitImage** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Refine1BitImage** [= *Boolean*]

【設定値】

値	説明
True	高精彩に表示する
False	そのまま表示する

【解説】
1 ビットイメージや 4,8 ビットグレースケールイメージを表示する際に有効になります。デフォルトは True です。
Refine1BitImage プロパティはイメージキットコントロールに表示するイメージが対象です。ただし、**Refine1BitImage** プロパティが True でも **Edit.EditEnable** プロパティが True(編集モード)の場合や **BackgroundImageFile** プロパティに有効なファイル名が設定されている場合には有効になりません(イメージはそのまま表示されます)。
このプロパティは IK7 との互換性保持のために残されていますので、使用しないことをお勧めいたします。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
IK7 と同じ動作にする場合は **StretchMode** プロパティに 0(vikRefineImage)を設定します。

RefineColorImage(イメージキットコントロール／カスタムプロパティ)

【機能】
24,32 ビットカラーイメージを縮小表示する際、高精彩に表示するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**RefineColorImage** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**RefineColorImage** [= *Boolean*]

【設定値】

値	説明
True	高精彩に表示する
False	そのまま表示する

【解説】
24,32 ビットカラーイメージを表示する際に有効になります。デフォルトは False です。
RefineColorImage プロパティはイメージキットコントロールに表示するイメージが対象です。ただし、**RefineColorImage** プロパティが True でも **Edit.EditEnable** プロパティが True(編集モード)の場合や **BackgroundImageFile** プロパティに有効なファイル名が設定されている場合には有効になりません(イメージはそのまま表示されます)。
このプロパティは IK7 との互換性保持のために残されていますので、使用しないことをお勧めいたします。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
IK7 と同じ動作にする場合は **StretchMode** プロパティに 0(vikRefineImage)を設定します。

ScrollBar(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロールに対してスクロールバーの表示の有無を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ScrollBar** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**ScrollBar** [= *Boolean*]

【設定値】

値	説明
True	イメージキットコントロールにスクロールバーを表示する
False	イメージキットコントロールにスクロールバーを表示しない

【解説】
デフォルトは False です。
ScrollBar プロパティを True にすることにより、イメージキットコントロール上に収まりきらないイメージを表示する際にスクロールバーを表示することができます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ScrollHeight,ScrollWidth（イメージキットコントロール／カスタムプロパティ）

【機能】
イメージキットコントロール上のイメージのスクロールの縦・横のピクセル数を取得または設定します。

【書式】
※ScrollHeight にて説明（ScrollWidth も同様な使い方）
(1)C++Builder　　*imagekitcontrolname*->ScrollHeight [= *int*]
(2)Delphi　　*imagekitcontrolname*.ScrollHeight [= *Integer*]

【設定値】
スクロールするピクセル数（1 以上）を設定します。デフォルトは 10 です。
ScrollHeight と ScrollWidth に両方 0 を設定した場合は、スクロールの機能は無効になります。

【解説】
ScrollHeight プロパティには縦方向にスクロールするピクセル数、ScrollWidth プロパティには横方向にスクロールするピクセル数を設定します。
ScrollImage メソッドを実行するとスクロールを実行することができます。上下スクロールの場合は ScrollHeight プロパティ（縦方向）の設定が、左右スクロールの場合は ScrollWidth プロパティ（横方向）の設定が、斜めスクロールの場合は、ScrollHeight プロパティ（縦方向）と ScrollWidth プロパティ（横方向）の設定がそれぞれ必要になります。
スクロール機能を使うことにより、プログラマは任意のボタンクリックなどのイベントと連動した、自由なイメージスクロールを実現できます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ShowBackgroundImage(イメージキットコントロール／カスタムプロパティ)

【機能】
イメージキットコントロールの背景に表示するイメージの表示/非表示を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ShowBackgroundImage** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**ShowBackgroundImage** [= *Boolean*]

【設定値】

値	説明
True	背景イメージを表示する
False	背景イメージを表示しない

【解説】
デフォルトは True です。
背景イメージとは **BackgroundImageFile** プロパティが指すイメージです。

【値の設定】 実行時

【値の参照】 実行時

ShowInDisp,ShowInMagnifier,ShowInPanWindow(イメージキットコントロール／カスタムプロパティ)

【機能】
基本イメージを表示するかどうかを取得または設定します。

【書式】
※**ShowInDisp** にて説明(その他も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->**ShowInDisp** [= *bool*]
(2)Delphi　　*imagekitcontrolname*.**ShowInDisp** [= *Boolean*]

【設定値】
ShowInDisp は基本イメージをイメージキットコントロールに表示するかどうかを設定します。
ShowInMagnifier は基本イメージを虫眼鏡ウィンドウに表示するかどうかを設定します。
ShowInPanWindow は基本イメージをパンウィンドウに表示するかどうかを設定します。

値	説明
True	基本イメージを表示する
False	基本イメージを表示しない

【解説】
デフォルトは True です。
基本イメージとは **ImageHandle** プロパティが指すイメージのことです。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

SimpleDispCtrl(イメージキットコントロール／カスタムプロパティ)

【機能】
基本イメージに限定して表示を行うかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**SimpleDispCtrl** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**SimpleDispCtrl** [= *Boolean*]

【設定値】

値	説明
True	基本イメージのみを表示する
False	背景イメージ・基本イメージ・階層イメージ、全てを表示する

【解説】
デフォルトは False です。
編集機能を使用しない場合(イメージ編集ツールバーや **ImageHdc** プロパティを使用しない)や背景イメージと階層イメージを使用しない場合、基本イメージに限定した表示を行うことにより描画速度を上げることができます。
True の場合、グリッド機能は使用できません(グリッドは無効)。
基本イメージとは **ImageHandle** プロパティが指すイメージのことです。

基本イメージに限定した表示を行うコード例:
(1)C++Builder
```
VImageKit1->LayerNo = -1;
VImageKit1->DisplayMode = vikActualSize;
VImageKit1->FileIO->FileName = "c:¥¥abc.jpg";
VImageKit1->FileIO->LoadFile(vikLoad);
VImageKit1->Edit->EditEnable = false;
VImageKit1->SimpleDispCtrl = true;
VImageKit1->DisplayImage();   //AutoDisplay が true の場合は不要
```
(2)Delphi
```
VImageKit1.LayerNo := -1;
VImageKit1.DisplayMode := vikActualSize;
VImageKit1.FileIO.FileName := 'c:¥abc.jpg';
VImageKit1.FileIO.LoadFile(vikLoad);
VImageKit1.Edit.EditEnable := False;
VImageKit1.SimpleDispCtrl := True;
VImageKit1.DisplayImage();   //AutoDisplay が True の場合は不要
```

【値の設定】 デザイン時、実行時

【値の参照】 実行時

StretchMode(イメージキットコントロール／カスタムプロパティ)

【機能】
ビットマップの伸縮モードを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**StretchMode** [= *short*]
(2)Delphi　*imagekitcontrolname*.**StretchMode** [= *Smallint*]

【設定値】

値	説明
0	ImageKit 独自の高精細表示を行います(**Refine1BitImage**,**RefineColorImage** プロパティ=True と同じ)。
1	論理 AND 演算子を使って、残す点の色と取り除く点の色を組み合わせます。 ビットマップがモノクロームのビットマップである場合、白のピクセルが消され、黒のピクセルが残ります。 (BLACKONWHITE)
2	論理 AND 演算子を使って、残す点の色と取り除く点の色を組み合わせます。 ビットマップがモノクロームである場合、黒のピクセルが消され、白のピクセルが残ります。 (WHITEONBLACK)
3	ピクセルを削除します。 取り除く点の情報を何らかの形で維持せようとはせず、単純にそれらの点を削除します。 (COLORONCOLOR)
4	コピー元長方形内のピクセルをコピー先長方形内のピクセルブロックに関連付けます。 コピー先ブロックの平均的な色は、コピー元のピクセルの色に近い色になります。 (HALFTONE)

()内の説明は WindowsAPI で使用する定数と同じ意味です。

【解説】
デフォルトは 3(COLORONCOLOR)です。プロパティの値が 1 から 4 の場合、WindowsAPI の SetStretchBltMode 関数の設定値と同じ動作となります。
1(BLACKONWHITE)と 2(WHITEONBLACK)は低画質ですが高速で、3(COLORONCOLOR)は実用的な画質で高速です。
4(HALFTONE)は高画質ですが低速です。
通常時は 3(COLORONCOLOR)を使用して、縮小時などに 4(HALFTONE)を使用するとよいでしょう。
0(vikRefineImage)で **Edit.EditEnable** プロパティが True(編集モード)の場合、高精細表示は有効になりません(イメージはそのまま表示されます)。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
IK7 と同じ動作にする場合は **StretchMode** プロパティに 0(vikRefineImage)を設定します。

【ImageKit8/9/10 ActiveX との違い】
定数の ikRefineImage が vikRefineImage に変更されました。それ以外は Windows ユニット(XE2 以降では先頭にユニット スコープ名が付加され、Winapi.Windows)で定義されている定数を使用します(ActiveX は ikRefineImage, ikBlackOnWhite, ikWhiteOnBlack, ikColorOnColor, ikHalftone)。

ToolTipXfromMouse,ToolTipYfromMouse,ToolTipTimeSecond(イメージキットコントロール／カスタムプロパティ)

【機能】
マウス座標値を表示するための情報を取得または設定します。

【書式】
※**ToolTipXfromMouse** にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**ToolTipXfromMouse** [= *short*]
(2)Delphi　*imagekitcontrolname*.**ToolTipXfromMouse** [= *Smallint*]

【設定値】
ToolTipXfromMouse はマウス位置からマウス座標値を表示する位置までのＸ方向のオフセット(ピクセル)。デフォルトは20。
ToolTipYfromMouse はマウス位置からマウス座標値を表示する位置までのＹ方向のオフセット(ピクセル)。デフォルトは10。
ToolTipTimeSecond はマウス座標値を表示する秒数(1～)。デフォルトは 3。

【解説】
ToolTip プロパティが vikNone 以外の場合に有効です。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Touch(イメージキットコントロール／カスタムプロパティ)

【機能】
ピンチイン、ピンチアウトによる拡大・縮小を有効にするかどうかを指定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Touch** [= *TTouchManager**]
(2)Delphi　*imagekitcontrolname*.**Touch** [= *TTouchManager*]

【設定値】
ピンチイン、ピンチアウトを有効にする場合は **Touch.InteractiveGestures** に **igZoom** を設定してください。
ピンチイン、ピンチアウトを無効にする場合は **Touch.InteractiveGestures** に **igZoom** を設定しないでください。

【解説】
タッチスクリーン対応アプリケーションで有効な機能です。
Touch プロパティの設定に関わらず、ベクトルデータの編集において、指でオブジェクトの操作を行うときは選択したオブジェクトのガイド表示の四角形の大きさが大きくなります。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
プロパティの名称が **EnbaleTouch** から変更されました。

TransBlue,TransGreen,TransRed(イメージキットコントロール／カスタムプロパティ)

【機能】
基本イメージ(ラスタ)の透過色を取得または設定します。

【書式】
※**TransBlue** にて説明(**TransGreen**,**TransRed** も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->**TransBlue** [= *short*]
(2)Delphi　　*imagekitcontrolname*.**TransBlue** [= *Smallint*]

【設定値】
TransBlue は透過色の青。
TransGreen は透過色の緑。
TransRed は透過色の赤。

0～255。

【解説】
基本イメージ(ラスタ)の一部を透過させる場合に使用します。設定した透過色の部分にはイメージキットコントロールの背面に配置されたコントロールが表示されます。
透過対象イメージは 8 ビットカラーもしくは 24 ビットカラーでなければなりません。
透過対象イメージは **ImageHandle** プロパティが指すラスタイメージのことです(ベクトルイメージが設定されている場合は無効です)。
透過色を設定する場合は、透過対象イメージよりも先に透過色を設定する必要があります。

透過色を白にする例:

```
VImageKit1.TransBlue := 255;
VImageKit1.TransGreen := 255;
VImageKit1.TransRed := 255;
VImageKit1.Transparent := True;
VImageKit1.LayerNo := -1;
VImageKit1.FileIO.LoadFile(vikLoad);
```

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Transparent(イメージキットコントロール／カスタムプロパティ)

【機能】
基本イメージ(ラスタ)の透過を有効にするかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Transparent** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Transparent** [= *Boolean*]

【設定値】

値	説明
True	透過を有効
False	透過を無効

【解説】
デフォルトは False です。
True の場合、**TransBlue**,**TransGreen**,**TransRed** プロパティで設定した色を透過色とすることができます。
透過対象イメージの **ImageHandle** プロパティがベクトルイメージの場合は無効です。

TransparentプロパティがTrueで**BackgroundImageFile**プロパティが空文字列(あるいは無効なファイル名)の場合
基本イメージの透過色の部分に下のウィンドウコントロールが表示されます。
イメージキットコントロールのサイズよりも大きなウィンドウコントロールの上に配置し、基本イメージを実寸ではなくスケールやストレッチで表示してください。

TransparentプロパティがTrueで**BackgroundImageFile**プロパティが有効なファイル名の場合
基本イメージの透過色の部分に背景イメージが表示されます。イメージキットコントロールの下のウィンドウコントロールは表示されません。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

VectorAntiAliasDisplay(イメージキットコントロール／カスタムプロパティ)

【機能】
ベクトルイメージのアンチエイリアス表示を有効にするかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**VectorAntiAliasDisplay** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**VectorAntiAliasDisplay** [= *Boolean*]

【設定値】

値	説明
True	アンチエイリアス表示を有効
False	アンチエイリアス表示を無効

【解説】
デフォルトは False です。
アンチエイリアス表示を有効にすると描画する時間は遅くなりますが、テキストや図形を綺麗に表示することができます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Xdpi,Ydpi(イメージキットコントロール／カスタムプロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの縦横の 1 インチあたりのピクセル数を取得または設定します。

【書式】
※Xdpi にて説明(Ydpi も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Xdpi** [= *int*]
(2)Delphi　*imagekitcontrolname*.**Xdpi** [= *Integer*]

【参照値】
Xdpi は横方向の 1 インチあたりのピクセル数。
Ydpi は縦方向の 1 インチあたりのピクセル数。

【解説】
ImageHandle プロパティに値が設定されると、Xdpi,Ydpi プロパティは自動的に更新されます。
また、基本イメージの解像度を変更する場合は LayerNo プロパティに-1 を設定してから SetDpi メソッドを実行します。

【値の設定】 実行時

【値の参照】 実行時

BitmapFromDib（イメージキットコントロール／カスタムメソッド）

【機能】
デバイス独立ビットマップ（DIB）からデバイス依存ビットマップ（DDB）に変換します。

【書式】
(1)C++Builder
[*HBITMAP =*]*imagekitcontrolname*->**BitmapFromDib**(NativeUInt ImageHandle)
[*HBITMAP =*]*imagekitcontrolname*->**BitmapFromDib**()
(2)Delphi
[*HBITMAP =*]*imagekitcontrolname*.**BitmapFromDib**(ImageHandle: THandle)
[*HBITMAP =*]*imagekitcontrolname*.**BitmapFromDib**

【引数】

名称	内容
ImageHandle	ラスタイメージのメモリハンドル

【戻り値】
ビットマップハンドル（実行に失敗した場合は 0 もしくは NULL）

【解説】
ラスタイメージが対象です（ベクトルイメージは不可）。

引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルが処理対象になります。
引数なしのメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されたメモリハンドルが処理対象になります。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

C++Builder の例：

```
// TBitmap にラスタイメージを設定
VImageKit1->LayerNo = -1;
if (VImageKit1->ImageHandle == 0) return;
if (VImageKit1->ImageKind != vikRasterImage) return;

Graphics::TBitmap *bm = new Graphics::TBitmap();
try
{
  bm->Handle = VImageKit1->BitmapFromDib();
  // イメージ処理
}
__finally
{
  delete bm;
}
```

Delphi の例：

```
{ TBitmap にラスタイメージを設定 }
var
  Bitmap1: TBitmap;
begin
  VImageKit1.LayerNo := -1;
  if VImageKit1.ImageHandle = 0 then Exit;
  if VImageKit1.ImageKind <> vikRasterImage then Exit;
```

```
    Bitmap1 := TBitmap.Create;
    try
      Bitmap1.Handle := VImageKit1.BitmapFromDib;
      { イメージ処理 }
    finally
      Bitmap1.Free;
    end;
  end;
```

BitBlt（イメージキットコントロール／カスタムメソッド）

【機能】
コピー元からコピー先のデバイスコンテキストへ、指定された矩形内の各ピクセルの色データをコピーします。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**BitBlt**(HDC hDCDst, int XDst, int YDst, int AWidth, int AHeight, HDC hDCSrc, int XSrc, int YSrc, unsigned dwRop)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**BitBlt**(hDCDst: HDC; XDst, YDst, AWidth, Aheight: Integer; hDCSrc: HDC; XSrc, Ysrc: Integer; dwRop: DWORD)

【引数】

名称	内容
hDCDst	コピー先のデバイスコンテキスト
XDst	コピー先矩形の左上隅の x 座標（ピクセル）
YDst	コピー先矩形の左上隅の y 座標（ピクセル）
AWidth	コピー先矩形の幅（ピクセル）
AHeight	コピー先矩形の高さ（ピクセル）
hDCSrc	コピー元のデバイスコンテキスト
XSrc	コピー元矩形の左上隅の x 座標（ピクセル）
YSrc	コピー元矩形の左上隅の y 座標（ピクセル）
dwRop	ラスタオペレーションコード

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ラスタオペレーションモードはデバイスコンテキストにどのようにコピーするかを指定します。

値	説明
0x00000042	コピー先の矩形を黒色で塗りつぶします。（BLACKNESS）
0x00550009	コピー先の矩形の色を反転します。（DSTINVERT）
0x00C000CA	論理 AND 演算子を使って、コピー元の色とコピー先の色を組み合わせます。（MERGECOPY）
0x00BB0226	論理 OR 演算子を使って、コピー元の色を反転した色と、コピー先の色を組み合わせます。（MERGEPAINT）
0x00330008	コピー元の色を反転して、コピー先へコピーします。（NOTSRCCOPY）
0x001100A6	論理 OR 演算子を使って、コピー元の色とコピー先の色を組み合わせ、さらに反転します。（NOTSRCERASE）
0x00F00021	指定したパターンをコピー先へコピーします。（PATCOPY）
0x005A0049	論理 XOR 演算子を使って、指定したパターンの色と、コピー先の色を組み合わせます。（PATINVERT）
0x00FB0A09	論理 OR 演算子を使って、指定したパターンの色と、コピー元の色を反転した色を組み合わせます。さらに論理 OR 演算子を使って、その結果と、コピー先の色を組み合わせます。（PATPAINT）
0x008800C6	論理 AND 演算子を使って、コピー元の色とコピー先の色を組み合わせます。（SRCAND）
0x00CC0020	コピー元の矩形をコピー先の矩形へそのままコピーします。（SRCCOPY）
0x00440328	論理 AND 演算子を使って、コピー先の色を反転した色と、コピー元の色を組み合わせます。（SRCERASE）
0x00660046	論理 XOR 演算子を使って、コピー元の色とコピー先の色を組み合わせます。（SRCINVERT）
0x00EE0086	論理 OR 演算子を使って、コピー元の色とコピー先の色を組み合わせます。（SRCPAINT）
0x00FF0062	コピー先の矩形を白色で塗りつぶします。（WHITENESS）

通常は 0x00CC0020(SRCCOPY)を指定します。

※16 進表記のため、Delphi は 0x を$に置き換えてください。

()内の説明はWindowsAPIで使用する定数と同じ意味です。

ClearClipBrd（イメージキットコントロール／カスタムメソッド）

【機能】
クリップボードをクリアします。

【書式】
(1)C++Builder　*imagekitcontrolname*->**ClearClipBrd**()
(2)Delphi　*imagekitcontrolname*.**ClearClipBrd**

【引数】
ありません。

【戻り値】
ありません。

【解説】
クリップボードをクリアします。

CopyImage(イメージキットコントロール／カスタムメソッド)

【機能】
メモリ上のイメージを別のメモリ上にコピーします。

【書式】
(1)C++Builder
[*NativeUInt =*]*imagekitcontrolname*->**CopyImage**(NativeUInt ImageHandle)
[*NativeUInt =*]*imagekitcontrolname*->**CopyImage**
(2)Delphi
[*THandle =*]*imagekitcontrolname*.**CopyImage**(ImageHandle: THandle)
[*THandle =*]*imagekitcontrolname*.**CopyImage**

【引数】

名称	内容
ImageHandle	コピー対象となるイメージのメモリハンドル

【戻り値】
成功の場合は新しいイメージのメモリハンドル、失敗の場合は 0 を返します。

【解説】
引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルが処理対象になります。
引数なしのメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されたメモリハンドルが処理対象になります。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

ラスタイメージは 1,4,8,16,24,32,48 ビットイメージが対象です。48 ビットイメージは **Scan** プロパティを使用してスキャンデバイスから 36,42,48 ビットカラーで取り込んだイメージのコピーなどに使用します。

CreateImage（イメージキットコントロール／カスタムメソッド）

【機能】
新規にラスタイメージを作成します。

【書式】
(1)C++Builder
[*NativeUInt =*]*imagekitcontrolname*->**CreateImage**(int AWidth, int AHeight, short BitCount, short Red, short Green, short Blue)
(2)Delphi
[*THandle =*]*imagekitcontrolname*.**CreateImage**(AWidth, AHeight: Integer; BitCount, Red, Green, Blue: Smallint)

【引数】

名称	内容
AWidth	作成するイメージの幅（ピクセル）
AHeight	作成するイメージの高さ（ピクセル）
BitCount	作成したいイメージのビット数(1,4,8,16,24,32,40,80) ※4 は 4 ビットカラー、40 は 4 ビットグレー、8 は 8 ビットカラー、80 は 8 ビットグレーとなります。
Red	作成するイメージカラーの赤（0～255）
Green	作成するイメージカラーの緑（0～255）
Blue	作成するイメージカラーの青（0～255）

【戻り値】
成功の場合は作成したラスタイメージのメモリハンドル、失敗の場合は 0 を返します。

【解説】
指定したビット数により、指定した RGB 値が有効にならない場合は、その値に一番近い値を割り当てます。（24 ビット以上のイメージの場合は指定した値がそのまま有効となります。）

DeleteBitmapObject（イメージキットコントロール／カスタムメソッド）

【機能】
ビットマップオブジェクトを削除します。

【書式】
(1)C++Builder　[*bool* =]*imagekitcontrolname*->**DeleteBitmapObject**(HBITMAP ABitmap)
(2)Delphi　[*Boolean* =]*imagekitcontrolname*.**DeleteBitmapObject**(ABitmap: HBITMAP)

【引数】

名称	内容
ABitmap	ビットマップハンドル

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
BitmapFromDib メソッドで取得したビットマップハンドルが不要になった場合に使用します。

DibFromBitmap（イメージキットコントロール／カスタムメソッド）

【機能】
デバイス依存ビットマップ（DDB）からデバイス独立ビットマップ（DIB）に変換します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**DibFromBitmap**(HBITMAP hBm, HPALETTE hPal)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**DibFromBitmap**(hBm: HBitmap; hPal: HPalette)

【引数】

名称	内容
hBm	ビットマップハンドル
hPal	ビットマップのパレット（パレットがない場合は 0）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
成功すると **LayerNo** プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）にラスタイメージのメモリハンドルが設定されます。
TBitmap からイメージを取得する場合などに使用します。引数として与えた hBm は削除されません。

C++Builder の例：

```
// TBitmap にラスタイメージを設定
Graphics::TBitmap *bm = new Graphics::TBitmap();
try
{
  bm->LoadFromFile("C:¥¥Bmp¥¥001.bmp");
  // イメージ処理 1
  VImageKit1->LayerNo = -1;
  VImageKit1->DibFromBitmap(bm->Handle, bm->Palette);
  // イメージ処理 2
}
__finally
{
  delete bm;
}
```

Delphi の例：

```
{ TBitmap からラスタイメージを取得 }
var
  Bitmap1: TBitmap;
begin
  Bitmap1 := TBitmap.Create;
  try
    Bitmap1.LoadFromFile('C:¥Bmp¥001.bmp');
    { イメージ処理 1 }
    VImageKit1.LayerNo := -1;
    VImageKit1.DibFromBitmap(Bitmap1.Handle, Bitmap1.Palette);
    { イメージ処理 2 }
  finally
    Bitmap1.Free;
  end;
end;
```

Display(イメージキットコントロール／カスタムメソッド)

【機能】
イメージ表示関連の動作を実行します。

【書式】
(1)C++Builder　[*bool* =]*imagekitcontrolname*->**Display**(TVIkDispMode DisplayMode)
(2)Delphi　[*Boolean* =]*imagekitcontrolname*.**Display**(DisplayMode: TVIkDispMode)

【TVIkDispMode 型】
ユニット
IkInit
type
TVIkDispMode = (vikClear, vikScale, vikStretch, vikActualSize, vikFitToWidth, vikFitToHeight);

【引数】

名称	内容
DisplayMode	表示モード vikClear: クリア vikScale: スケール(イメージのサイズがコントロールのサイズより大きい場合は縮小して表示) vikStretch: ストレッチ(コントロールのサイズに合わせてイメージをはめこむ) vikActualSize: 実寸(**DispScaleX**,**DispScaleY** プロパティによる) vikFitToWidth: 横幅に合わせる(コントロールの **Width** プロパティに合わせてイメージをはめこむ) vikFitToHeight: 縦幅に合わせる(コントロールの **Height** プロパティに合わせてイメージをはめこむ)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
DisplayMode が vikClear の場合はイメージキットコントロールの描画領域に表示されているイメージを消去します。再度 Display メソッドを実行するまで有効です。
DisplayMode が vikClear 以外の場合は **ShowInDisp** プロパティが True に設定されているイメージを表示します(基本イメージ、**Layer** イメージが対象)。
イメージを表示する順番は、基本イメージ、**Layer** イメージ(インデックスが小さい方から大きい方へ)となります。
ただし、基本イメージが設定されていない場合、**Layer** イメージは表示されません。
このメソッドは IK7 との互換性保持のために残されていますので、代わりに DisplayImage メソッドを使用することをお勧めいたします。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikClear, ikScale, ikStretch, ikActualSize, ikFitToWidth, ikFitToHeight)。

DisplayImage（イメージキットコントロール／カスタムメソッド）

【機能】
イメージ表示関連の動作を実行します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**DisplayImage**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**DisplayImage**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
DisplayImage メソッドを実行する前に **DisplayMode** プロパティを設定してください。
DisplayMode プロパティが vikClear の場合はイメージキットコントロールの描画領域に表示されているイメージを消去します。再度 **DisplayImage** メソッドを実行するまで有効です。
DisplayMode プロパティが vikClear 以外の場合は **ShowInDisp** プロパティが True に設定されているイメージを表示します（基本イメージ、**Layer** イメージが対象）。
イメージを表示する順番は、基本イメージ、**Layer** イメージ（インデックスが小さい方から大きい方へ）となります。
ただし、基本イメージが設定されていない場合、**Layer** イメージは表示されません。

FreeMemory(イメージキットコントロール／カスタムメソッド)

【機能】
イメージのメモリを解放します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**FreeMemory**(NativeUInt ImageHandle)
[*bool* =]*imagekitcontrolname*->**FreeMemory**()
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**FreeMemory**(ImageHandle: THandle)
[*Boolean* =]*imagekitcontrolname*.**FreeMemory**

【引数】

名称	内容
ImageHandle	解放するイメージのメモリハンドル

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ラスタイメージとベクトルイメージおよび Raw データが対象です。

引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルが処理対象になります。
引数なしのメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されたメモリハンドルが処理対象になります。成功するとメモリを解放し、プロパティに 0 を設定します。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

また、**ImageHandle** プロパティや **Layer[Index].ImageHandle** プロパティについては明示的に解放しなくても、再設定時やコントロールが破棄される時に自動的に解放されます(Index は 0～99)。
なお、明示的に解放する場合はメモリの 2 重解放を防ぐため、メソッド実行後にプロパティ値に 0 を設定するようにしてください。
Delphi の例:
(1)
```
VImageKit1.LayerNo := -1;
VImageKit1.FreeMemory;
VImageKit1.ImageHandle := 0;
```
(2)
```
VImageKit1.FreeMemory(VImageKit1.ImageHandle);
VImageKit1.ImageHandle := 0;
```
(3)
```
VImageKit1.ImageHandle := 0;
```

※(3)は FreeMemory メソッドを使用しておりませんが、(1)(2)と同じくメモリを解放します。

GetDpi(イメージキットコントロール／カスタムメソッド)

【機能】
イメージの解像度を取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**GetDpi**()
[*bool =*]*imagekitcontrolname*->**GetDpi**(float &Xdpi, float &Ydpi)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**GetDpi**
[*Boolean =*]*imagekitcontrolname*.**GetDpi**(var Xdpi, Ydpi: Single)

【引数】

名称	内容
Xdpi	取得するX(横)方向の1インチあたりのピクセル数
Ydpi	取得するY(縦)方向の1インチあたりのピクセル数

【戻り値】
成功の場合はTrue、失敗の場合はFalseを返します。

【解説】
LayerNoプロパティが示すプロパティ(**ImageHandle**プロパティもしくは**Layer[LayerNo].ImageHandle**プロパティ)のイメージの解像度を取得します。

引数なしのメソッド
成功すると**ImageHandle**プロパティであれば**Xdpi**,**Ydpi**プロパティに、**Layer[LayerNo].ImageHandle**プロパティであれば**Layer[LayerNo].Xdpi**,**Layer[LayerNo].Ydpi**プロパティに値が設定されます。
このメソッドを実行しなくても**ImageHandle**プロパティや**Layer[LayerNo].ImageHandle**プロパティに値が設定された段階で解像度は自動的に設定されます。
引数ありのメソッド
成功すると引数Xdpi,Ydpiに値が設定されます。

メソッドの引数ありなしによる違いは取得する解像度が整数型か浮動小数点型かということです。

【ImageKit7/8/9/10 ActiveXとの違い】
ActiveXの**GetDpiF**メソッドは引数ありの**GetDpi**メソッドと同じです。

GetDpiFromHdc（イメージキットコントロール／カスタムメソッド）

【機能】
デバイスコンテキストの解像度を取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**GetDpiFromHdc**(HDC DC, int &Xdpi, int &Ydpi)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**GetDpiFromHdc**(DC: HDC; var Xdpi, Ydpi: Integer)

【引数】

名称	内容
DC	デバイスコンテキスト
Xdpi	取得するX（横）方向の1インチあたりのピクセル数
Ydpi	取得するY（縦）方向の1インチあたりのピクセル数

【戻り値】
成功の場合はTrue、失敗の場合はFalseを返します。

【解説】
スクリーンやプリンタの解像度を取得できます。

GetFromClipBrd(イメージキットコントロール／カスタムメソッド)

【機能】
クリップボードからラスタイメージやベクトルイメージを取り込みます。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**GetFromClipBrd**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**GetFromClipBrd**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
成功すると **LayerNo** プロパティが示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)にラスタイメージやベクトルイメージのメモリハンドルが設定されます。
取得するイメージがベクトルの場合、EMF 形式として扱います。

GetImageType（イメージキットコントロール／カスタムメソッド）

【機能】
イメージの各種情報をプロパティに設定します。

【書式】
(1)C++Builder　[*bool* =]*imagekitcontrolname*->**GetImageType**()
(2)Delphi　[*Boolean* =]*imagekitcontrolname*.**GetImageType**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
LayerNo プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）のイメージの各種情報を取得します。成功すると次のプロパティに値が設定されます。

ImageHandleプロパティが対象の場合
BitCount,Gray,ImageKind,ImageSize,ImageHeight,ImageWidth,ImageWidthByte,Mask1632,PalCount,Xdpi,Ydpi
Layer[LayerNo].ImageHandleプロパティが対象の場合
Layer[LayerNo].BitCount,Layer[LayerNo].Gray,Layer[LayerNo].Height,Layer[LayerNo].ImageKind,Layer[LayerNo].ImageSize,Layer[LayerNo].Mask1632,Layer[LayerNo].PalCount,Layer[LayerNo].Width,Layer[LayerNo].WidthByte,Layer[LayerNo].Xdpi,Layer[LayerNo].Ydpi

このメソッドを実行しなくても **ImageHandle** プロパティや **Layer[LayerNo].ImageHandle** プロパティに値が設定された段階で各種情報は自動的に設定されます。
設定されるプロパティについては、各プロパティの説明を参照してください。

GetMemorySize（イメージキットコントロール／カスタムメソッド）

【機能】
イメージの使用しているメモリサイズを取得します。

【書式】
(1)C++Builder
[*int =*]*imagekitcontrolname*->**GetMemorySize**(NativeUInt ImageHandle)
[*int =*]*imagekitcontrolname*->**GetMemorySize**()
(2)Delphi
[*Integer =*]*imagekitcontrolname*.**GetMemorySize**(ImageHandle: THandle)
[*Integer =*]*imagekitcontrolname*.**GetMemorySize**

【引数】

名称	内容
ImageHandle	イメージのメモリハンドル

【戻り値】
使用しているメモリサイズ（バイト単位）を返します。（失敗した場合は 0）

【解説】
引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルが処理対象になります。
引数なしのメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されたメモリハンドルが処理対象になります。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

GetOneBitPalCount（イメージキットコントロール／カスタムメソッド）

【機能】
1 ビットカラーイメージのパレット 0 とパレット 1 のピクセル数を取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**GetOneBitPalCount**(int &Pal0, int &Pal1, NativeUInt ImageHandle)
[*bool =*]*imagekitcontrolname*->**GetOneBitPalCount**(int &Pal0, int &Pal1)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**GetOneBitPalCount**(var Pal0, Pal1: Integer; ImageHandle: THandle)
[*Boolean =*]*imagekitcontrolname*.**GetOneBitPalCount**(var Pal0, Pal1: Integer)

【引数】

名称	内容
Pal0	取得する 1 ビットカラーイメージのパレット 0 のピクセル数
Pal1	取得する 1 ビットカラーイメージのパレット 1 のピクセル数
ImageHandle	ラスタイメージのメモリハンドル

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
1 ビットカラーイメージが対象です。

引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルが処理対象になります。
ImageHandleが不要なメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されたメモリハンドルが処理対象になります。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

パレットの色については、**GetPaletteFromImage** メソッドを参照してください。

GetPaletteFromImage（イメージキットコントロール／カスタムメソッド）

【機能】
ラスタイメージのパレット情報を取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**GetPaletteFromImage**(short * Red, const int Red_Size, short * Green, const int Green_Size, short * Blue, const int Blue_Size, NativeUInt ImageHandle)
[*bool =*]*imagekitcontrolname*->**GetPaletteFromImage**(short * Red, const int Red_Size, short * Green, const int Green_Size, short * Blue, const int Blue_Size)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**GetPaletteFromImage**(var Red, Green, Blue: array of Smallint; ImageHandle: THandle)
[*Boolean =*]*imagekitcontrolname*.**GetPaletteFromImage**(var Red, Green, Blue: array of Smallint)

【引数】

名称	内容
Red	取得する赤のパレット情報の配列
Green	取得する緑のパレット情報の配列
Blue	取得する青のパレット情報の配列

※C++Builder の場合は、配列の要素数-1 を Red_Size,Green_Size,Blue_Size に与えます。

ImageHandle	ラスタイメージのメモリハンドル

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルが処理対象になります。
ImageHandleが不要なメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されたメモリハンドルが処理対象になります。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

パレット情報を取得する RGB のそれぞれの配列の数は **PalCount** プロパティの値になります。
PalCount プロパティは **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに値が設定された段階で更新されます。また、**GetImageType** メソッドでも取得できます。

コード例：
(1)C++Builder

```
VImageKit1->LayerNo = -1;
if (VImageKit1->PalCount < 1) return;
short *r = new short[VImageKit1->PalCount];
short *g = new short[VImageKit1->PalCount];
short *b = new short[VImageKit1->PalCount];
VImageKit1->GetPaletteFromImage(r, VImageKit1->PalCount - 1, g, VImageKit1->PalCount - 1, b,
VImageKit1->PalCount - 1);
// r, g, b を使用した処理
VImageKit1->SetPaletteToImage(r, VImageKit1->PalCount - 1, g, VImageKit1->PalCount - 1, b,
VImageKit1->PalCount - 1);

delete[] r;
delete[] g;
```

```
    delete[] b;
```

(2)Delphi

```
    var
      r, g, b: array of Smallint;
    begin
      VImageKit1.LayerNo := -1;
      if VImageKit1.PalCount < 1 then Exit;
      SetLength(r, VImageKit1.PalCount);
      SetLength(g, VImageKit1.PalCount);
      SetLength(b, VImageKit1.PalCount);
      VImageKit1.GetPaletteFromImage(r, g, b);
      { r, g, b を使用した処理 }
      VImageKit1.SetPaletteToImage(r, g, b);
    end;
```

【ImageKit7/8/9/10 ActiveX との違い】

メソッドの名称が **GetPalette** から変更されました。

GetSystemPalette(イメージキットコントロール／カスタムメソッド)

【機能】
現在の PC のシステムパレットを取得します。

【書式】
(1)C++Builder　[*short =*]*imagekitcontrolname*->**GetSystemPalette**()
(2)Delphi　[*Smallint =*]*imagekitcontrolname*.**GetSystemPalette**

【引数】
ありません。

【戻り値】

	内容
	現在のコンピュータのシステムパレットを返します。 2:2 色(1 ビットカラー) 16:16 色(4 ビットカラー) 256:256 色(8 ビットカラー) 0:16 ビットカラー、24 ビットカラー、32 ビットカラー

【解説】
現在のコンピュータのシステムパレットを取得します。

IsClipBrdData（イメージキットコントロール／カスタムメソッド）

【機能】
クリップボードにデータがあるかどうかをチェックします。

【書式】
(1)C++Builder　[*bool* =]*imagekitcontrolname*->**IsClipBrdData**()
(2)Delphi　[*Boolean* =]*imagekitcontrolname*.**IsClipBrdData**

【引数】
ありません。

【戻り値】
データがある場合は True、データがない場合は False が返されます。

【解説】
クリップボードにデータがあるかどうかをチェックします。

Refresh（イメージキットコントロール／カスタムメソッド）

【機能】
イメージキットコントロールの再描画を実行します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Refresh**()
(2)Delphi　*imagekitcontrolname*.**Refresh**

【引数】
ありません。

【戻り値】
ありません。

【解説】
表示しているイメージ（メモリハンドルやメモリデバイスコンテキスト）に対して処理を行った後などに実行すると、その結果が反映されます。

ScrollImage(イメージキットコントロール／カスタムメソッド)

【機能】
ScrollHeight,**ScrollWidth** プロパティで設定した条件に応じてイメージをスクロールさせます。

【書式】
(1)C++Builder　[*bool* =]*imagekitcontrolname*->**ScrollImage**(TVIkScrollDir Dir)
(2)Delphi　[*Boolean* =]*imagekitcontrolname*.**ScrollImage**(Dir: TVIkScrollDir)

【TVIkScrollDir 型】
ユニット
IkInit
type
TVIkScrollDir = (vikUpLeft, vikUp, vikUpRight, vikRight, vikDownRight, vikDown, vikDownLeft, vikLeft);

【引数】

名称	内容	
Dir	スクロールする方向	
	vikUpLeft:	左上
	vikUp:	上
	vikUpRight:	右上
	vikRight:	右
	vikDownRight:	右下
	vikDown:	下
	vikDownLeft:	左下
	vikLeft:	左

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドで実行させる動作は、予め **ScrollHeight**,**ScrollWidth** プロパティに適切な値が設定されている必要があります。
また、実寸表示以外の場合は無効です。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikUpLeft, ikUp, ikUpRight, ikRight, ikDownRight, ikDown, ikDownLeft, ikLeft)。

SetDpi(イメージキットコントロール／カスタムメソッド)

【機能】
イメージの解像度を新たに設定します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**SetDpi**()
[*bool =*]*imagekitcontrolname*->**SetDpi**(float Xdpi, float Ydpi)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**SetDpi**
[*Boolean =*]*imagekitcontrolname*.**SetDpi**(Xdpi, Ydpi: Single)

【引数】

名称	内容
Xdpi	設定するX(横)方向の1インチあたりのピクセル数
Ydpi	設定するY(縦)方向の1インチあたりのピクセル数

【戻り値】
成功の場合はTrue、失敗の場合はFalseを返します。

【解説】
LayerNoプロパティが示すプロパティ(**ImageHandle**プロパティもしくは**Layer[LayerNo].ImageHandle**プロパティ)のイメージに解像度を設定します。

引数なしのメソッド
成功すると**ImageHandle**プロパティであれば**Xdpi**,**Ydpi**プロパティの値が、**Layer[LayerNo].ImageHandle**プロパティであれば**Layer[LayerNo].Xdpi**,**Layer[LayerNo].Ydpi**プロパティの値がイメージに対して設定されます。
引数ありのメソッド
成功すると引数Xdpi,Ydpiの値が、**LayerNo**プロパティが示すイメージに対して設定されます。

【ImageKit7/8/9/10 ActiveXとの違い】
ActiveXの**SetDpiF**メソッドは引数ありの**SetDpi**メソッドと同じです。

SetPaletteToImage（イメージキットコントロール／カスタムメソッド）

【機能】
ラスタイメージのパレット情報を設定します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**SetPaletteToImage**(const short * Red, const int Red_Size, const short * Green, const int Green_Size, const short * Blue, const int Blue_Size, NativeUInt ImageHandle)
[*bool =*]*imagekitcontrolname*->**SetPaletteToImage**(const short * Red, const int Red_Size, const short * Green, const int Green_Size, const short * Blue, const int Blue_Size)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**SetPaletteToImage**(const Red, Green, Blue: array of Smallint; ImageHandle: THandle)
[*Boolean =*]*imagekitcontrolname*.**SetPaletteToImage**(const Red, Green, Blue: array of Smallint)

【引数】

名称	内容
Red	設定する赤のパレット情報の配列
Green	設定する緑のパレット情報の配列
Blue	設定する青のパレット情報の配列

※C++Builder の場合、配列の要素数-1 を Red_Size, Green_Size, Blue_Size に与えます。

ImageHandle	ラスタイメージのメモリハンドル

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルが処理対象になります。
ImageHandleが不要なメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されたメモリハンドルが処理対象になります。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

設定できるパレット数は、イメージのパレット数と同じでなければなりません。
パレット情報を設定する RGB のそれぞれの配列の数は **PalCount** プロパティの値になります。
PalCount プロパティは **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに値が設定された段階で更新されます。また、**GetImageType** メソッドでも取得できます。

【ImageKit7/8/9/10 ActiveX との違い】
メソッドの名称が **SetPalette** から変更されました。

SetToClipBrd(イメージキットコントロール／カスタムメソッド)

【機能】
ラスタイメージやベクトルイメージをクリップボードへコピーします。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**SetToClipBrd**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**SetToClipBrd**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
成功すると**LayerNo** プロパティが示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)のイメージをクリップボードへコピーします。
対象となるイメージがベクトルの場合、EMF 形式でクリップボードにコピーします。

StretchBlt（イメージキットコントロール／カスタムメソッド）

【機能】

コピー元からコピー先のデバイスコンテキストへ、指定された矩形内の各ピクセルの色データをコピーします。

【書式】

(1)C++Builder

[*bool =*]*imagekitcontrolname*->**StretchBlt**(HDC DCDst, int XDst, int YDst, int WidthDst, int HeightDst, HDC DCSrc, int XSrc, int YSrc, int WidthSrc, int HeightSrc, unsigned dwRop)

(2)Delphi

[*Boolean =*]*imagekitcontrolname*.**StretchBlt**(DCDst: HDC; XDst, YDst, WidthDst, HeightDst: Integer; DCSrc: HDC; XSrc, YSrc, WidthSrc, HeightSrc: Integer; dwRop: DWORD)

【引数】

名称	内容
DCDst	コピー先のデバイスコンテキスト
XDst	コピー先矩形の左上隅の x 座標（ピクセル）
YDst	コピー先矩形の左上隅の y 座標（ピクセル）
WidthDst	コピー先矩形の幅（ピクセル）
HeightDst	コピー先矩形の高さ（ピクセル）
DCSrc	コピー元のデバイスコンテキスト
XSrc	コピー元矩形の左上隅の x 座標（ピクセル）
YSrc	コピー元矩形の左上隅の y 座標（ピクセル）
WidthSrc	コピー元矩形の幅（ピクセル）
HeightSrc	コピー元矩形の高さ（ピクセル）
dwRop	ラスタオペレーションコード

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

コピー元のイメージをメモリ内で拡大または縮小し、その結果をコピー先の矩形へコピーします。拡大または縮小を行った後で、パターンまたはコピー先ピクセルの色データを組み合わせます。

WidthSrc と WidthDest の各パラメータの符号、または HeightSrc と HeightDest の各パラメータの符号が異なる場合、ミラーイメージを作成します。WidthSrc と WidthDest の符号が異なる場合、x 軸を中心にしてミラーイメージを作成します。HeightSrc と HeightDest の符号が異なる場合、y 軸を中心にしてミラーイメージを作成します。

値	説明
0x00000042	コピー先の矩形を黒色で塗りつぶします。（BLACKNESS）
0x00550009	コピー先の矩形の色を反転します。（DSTINVERT）
0x00C000CA	論理 AND 演算子を使って、コピー元の色とコピー先の色を組み合わせます。（MERGECOPY）
0x00BB0226	論理 OR 演算子を使って、コピー元の色を反転した色と、コピー先の色を組み合わせます。（MERGEPAINT）
0x00330008	コピー元の色を反転して、コピー先へコピーします。（NOTSRCCOPY）
0x001100A6	論理 OR 演算子を使って、コピー元の色とコピー先の色を組み合わせ、さらに反転します。（NOTSRCERASE）
0x00F00021	指定したパターンをコピー先へコピーします。（PATCOPY）
0x005A0049	論理 XOR 演算子を使って、指定したパターンの色と、コピー先の色を組み合わせます。（PATINVERT）
0x00FB0A09	論理 OR 演算子を使って、指定したパターンの色と、コピー元の色を反転した色を組み合わせます。さらに論理 OR 演算子を使って、その結果と、コピー先の色を組み合わせます。（PATPAINT）
0x008800C6	論理 AND 演算子を使って、コピー元の色とコピー先の色を組み合わせます。（SRCAND）
0x00CC0020	コピー元の矩形をコピー先の矩形へそのままコピーします。（SRCCOPY）
0x00440328	論理 AND 演算子を使って、コピー先の色を反転した色と、コピー元の色を組み合わせます。（SRCERASE）
0x00660046	論理 XOR 演算子を使って、コピー元の色とコピー先の色を組み合わせます。（SRCINVERT）

0x00EE0086　　論理 OR 演算子を使って、コピー元の色とコピー先の色を組み合わせます。（SRCPAINT）
0x00FF0062　　コピー先の矩形を白色で塗りつぶします。（WHITENESS）

通常は 0x00CC0020(SRCCOPY)を指定します。

※16 進表記のため、Delphi は 0x を$に置き換えてください。
（）内の説明は WindowsAPI で使用する定数と同じ意味です。

Zoom（イメージキットコントロール／カスタムメソッド）

【機能】
指定した矩形領域を拡大もしくは縮小します。

【書式】
(1)C++Builder　[*bool* =]*imagekitcontrolname*->**Zoom**(int ALeft, int ATop, int ARight, int ABottom)
(2)Delphi　[*Boolean* =]*imagekitcontrolname*.**Zoom**(ALeft, ATop, ARight, ABottom: Integer)

【引数】

名称	内容
ALeft	対象となる矩形の左位置（ピクセル）
ATop	対象となる矩形の上位置（ピクセル）
ARight	対象となる矩形の右位置（ピクセル）
ABottom	対象となる矩形の下位置（ピクセル）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
イメージキットコントロールのサイズに合わせて、指定した矩形内でアスペクト比を保持しながら拡大もしくは縮小します。また、実寸表示以外の場合は無効です。メソッド実行後、**DispScaleX**,**DispScaleY** プロパティは更新されます。

AfterScan(イメージキットコントロール／カスタムイベント)

【機能】
スキャンデバイスからイメージを 1 枚取り込んだ後に発生します。

【書式】
(1)C++Builder
*imagekitcontrolname***AfterScan**(TObject *Sender, NativeUInt DibHandle, NativeUInt OrgHandle, int ImageCount, short BitOrder)
(2)Delphi
*imagekitcontrolname***AfterScan**(Sender: TObject; DibHandle, OrgHandle: THandle; ImageCount: Integer; BitOrder: Smallint)

【引数】

名称	内容
DibHandle	Windows で処理できる DIB のメモリハンドル
OrgHandle	Windows で処理できないメモリハンドル (12,14,16 ビットグレースケールや 36,42,48 ビットカラー、および圧縮モード時)
ImageCount	取り込んだイメージの数
BitOrder	取り込んだイメージのビットがバイトの左端から始まっているか、右端から始まっているかを表します。 0:右端から(LSB)、1:左端から(MSB)

【解説】
取り込んだイメージのメモリハンドルはイベント終了後に解放されますので(DibHandle と OrgHandle の両方)、メモリハンドルを残しておきたい場合は、メモリをコピーするかファイルへ保存してください。
(1)ネイティブ転送、メモリ転送で非圧縮モードの場合
取り込んだイメージが 1,4,8,24 ビットイメージの場合は、イメージを DibHandle に設定し OrgHandle は 0 になります。
取り込んだイメージが 12,14,16 ビットグレースケールもしくは 36,42,48 ビットカラーの場合は、イメージを OrgHandle に設定し、OrgHandle を 8 ビットグレースケールもしくは 24 ビットカラーに減色したイメージを DibHandle に設定します。ただし、DibHandle 用のメモリが確保できない場合は 0 となります。
(2)メモリ転送で圧縮モードの場合
DibHandle は 0 で OrgHandle に取り込んだイメージを設定します。
(3)ファイル転送の場合
DibHandle,OrgHandle とも 0 になります。

Scan.UiMode プロパティを vikScanNONUI、かつ **Scan.UnitMode** プロパティを vikScanPixel に設定すると、DibHandle と OrgHandle の 2 つのメモリハンドルの解像度情報が 1 になる場合があります。
取り込み処理を中止する場合は、イベントの中で **Scan.Cancel** プロパティを True に設定します。

BeforeScan(イメージキットコントロール／カスタムイベント)

【機能】
スキャンデバイスからイメージを 1 枚取り込む前に発生します。

【書式】
(1)C++Builder　*imagekitcontrolname*BeforeScan(TObject *Sender, int AWidth, int AHeight, int ImageCount)
(2)Delphi　*imagekitcontrolname*BeforeScan(Sender: TObject; AWidth, AHeight, ImageCount: Integer)

【引数】

名称	内容
AWidth	転送されるイメージの幅(ピクセル)
AHeight	転送されるイメージの高さ(ピクセル)
ImageCount	取り込もうとするイメージの順番(1～)

【解説】
イベント内で実際に転送されるイメージの情報を Scan.XResolution,Scan.YResolution,Scan.BitDepth,Scan.PixelType, Scan.Compression プロパティで参照できます。
Scan.PaperSize プロパティに 1000(vikScanUndefinedSize)以上を設定すると AWidth と AHeight が-1 になる場合があります。また、実際の原稿サイズよりも AWidth,AHeight とも大きくなる場合があります。
Scan.PixelType プロパティに vikScanPixelAutomation を設定すると BitDepth と PixelType が実際に取り込んだ画像と異なる場合があります。
取り込み処理を中止する場合は、イベントの中で Scan.Cancel プロパティを True に設定します。

Scan.XResolution,Scan.YResolution
TWAIN の仕様では取り込み単位(インチや cm など)に依存した値になりますが、データソースによっては取り込み単位に関わらず、常に DPI となるものがあります。また、ピクセル単位で取り込むと 1 が設定される場合があります。
Scan.BitDepth
12,14 ビットグレースケールは 16 となり、36,42 ビットカラーは 48 となります。
Scan.Compression
ファイル転送およびメモリ転送の際に使用されます。(ネイティブ転送では意味を持ちません。)

ClickOnPanWindow（イメージキットコントロール／カスタムイベント）

【機能】
パンウィンドウ表示中にその領域内でマウスの左ボタンを押した際に発生します。

【書式】
(1)C++Builder　*imagekitcontrolname***ClickOnPanWindow**(TObject *Sender, int X, int Y)
(2)Delphi　*imagekitcontrolname***ClickOnPanWindow**(Sender: TObject; X, Y: Integer)

【引数】

名称	内容
X, Y	パンウィンドウのマウスクリック座標（ピクセル）

EditToolBar(イメージキットコントロール／カスタムイベント)

【機能】
イメージ編集ツールバーを表示もしくは非表示にすると発生します。

【書式】
(1)C++Builder　*imagekitcontrolname***EditToolBar**(TObject *Sender, TVIkEditToolBarMode Mode)
(2)Delphi　*imagekitcontrolname***EditToolBar**(Sender: TObject; Mode: TVIkEditToolBarMode)

【TVIkEditToolBarMode 型】
ユニット
IkInit
type
TVIkEditToolBarMode = (vikHide, vikRaster, vikVector);

【引数】

名称	内容
Mode	表示モード vikHide:　非表示 vikRaster:ラスタイメージ用 vikVector:ベクトルイメージ用

【解説】
イメージ編集ツールバーは **Edit** プロパティの **ShowToolBar** メソッドを実行することにより表示(非表示)できます。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikHide, ikRaster, ikVector)。

EndDispImage(イメージキットコントロール／カスタムイベント)

【機能】
イメージを表示した後に発生します。

【書式】
(1)C++Builder
*imagekitcontrolname***EndDispImage**(TObject *Sender, int OriginX, int OriginY, int ALeft, int ATop, int ARight, int ABottom, double ScaleWidth, double ScaleHeight)
(2)Delphi
*imagekitcontrolname***EndDispImage**(Sender: TObject; OriginX, OriginY, ALeft, ATop, ARight, ABottom: Integer; ScaleWidth, ScaleHeight: Double)

【引数】

名称	内容
OriginX	イメージキットコントロール内でのイメージの表示開始座標(X)
OriginY	イメージキットコントロール内でのイメージの表示開始座標(Y)
ALeft	表示されている実イメージの原点からの位置(左)
ATop	表示されている実イメージの原点からの位置(上)
ARight	表示されている実イメージの原点からの位置(右)
ABottom	表示されている実イメージの原点からの位置(下)
ScaleWidth	実イメージと表示イメージとの横方向のスケール(比率)
ScaleHeight	実イメージと表示イメージとの縦方向のスケール(比率)

座標や位置を表す単位はピクセルとなります。

【解説】
イメージを表示した直後に特定の処理を実行する場合は、イベント内でコードを記述します。
OriginX,OriginY は **BorderVisible**,**Appearance**,**Grad** プロパティの設定値により異なります。

EndLoad,StartLoad(イメージキットコントロール／カスタムイベント)

【機能】

EndLoad　　ファイルの読込処理後に発生します。
StartLoad　　ファイルの読込処理前に発生します。

【書式】

※**EndLoad** にて説明(**StartLoad** も同様な使い方)

(1)C++Builder

*imagekitcontrolname***EndLoad**(TObject *Sender, const UnicodeString FileName)

(2)Delphi

*imagekitcontrolname***EndLoad**(Sender: TObject; const FileName: string)

【引数】

名称	内容
FileName	読込ファイル名

【解説】

File プロパティの **FileLoadAsRawData**,**LoadFile(Mem)**メソッドを実行した場合に発生します。
LoadFileMem メソッドでは Raw データからの読込のため FileName は空文字列となります。

【ImageKit7/8/9/10 ActiveX との違い】

・C++Builder で使用する場合、引数の FileName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の FileName が string 型に変更されました。

EndPopUpMenu（イメージキットコントロール／カスタムイベント）

【機能】
ポップアップメニューの処理が終了した直後に発生します。

【書式】
(1)C++Builder　*imagekitcontrolname*EndPopUpMenu(TObject *Sender)
(2)Delphi　*imagekitcontrolname*EndPopUpMenu(Sender: TObject)

【引数】
ありません。

【解説】
ポップアップメニューとは、イメージ編集ツールバーを表示した状態でマウスを右クリックして表示されるメニューリストのことです。

EndRasterToVector,StartRasterToVector(イメージキットコントロール／カスタムイベント)

【機能】

EndRasterToVector	ベクトル化(ラスタからベクトルへの変換)処理が終了した後で発生します。
StartRasterToVector	ベクトル化(ラスタからベクトルへの変換)処理を開始する前に発生します。

【書式】

※EndRasterToVector にて説明(StartRasterToVector も同様な使い方)

(1)C++Builder　*imagekitcontrolname*EndRasterToVector(TObject *Sender)

(2)Delphi　*imagekitcontrolname*EndRasterToVector(Sender: TObject)

【引数】

ありません。

【解説】

Vector プロパティの RasterToVector メソッドを実行した場合に発生します。

EndSave,StartSave(イメージキットコントロール／カスタムイベント)

【機能】

EndSave	ファイルの保存処理後に発生します。
StartSave	ファイルの保存処理前に発生します。

【書式】

※**EndSave** にて説明(**StartSave** も同様な使い方)

(1)C++Builder

*imagekitcontrolname***EndSave**(TObject *Sender, const UnicodeString FileName)

(2)Delphi

*imagekitcontrolname***EndSave**(Sender: TObject; const FileName: string)

【引数】

名称	内容
FileName	保存ファイル名

【解説】

File プロパティの **FileSaveAsRawData,SaveFile(Mem)**メソッドを実行した場合に発生します。

SaveFileMem メソッドでは Raw データへ保存するため FileName は空文字列となります。

【ImageKit7/8/9/10 ActiveX との違い】

・C++Builder で使用する場合、引数の FileName が UnicodeString 型に変更されました。

・Delphi で使用する場合、引数の FileName が string 型に変更されました。

EndVectorToRaster,StartVectorToRaster(イメージキットコントロール／カスタムイベント)

【機能】

EndVectorToRaster	ラスタ化(ベクトルからラスタへの変換)処理が終了した後で発生します。
StartVectorToRaster	ラスタ化(ベクトルからラスタへの変換)処理を開始する前に発生します。

【書式】

※EndVectorToRaster にて説明(StartVectorToRaster も同様な使い方)

(1)C++Builder　*imagekitcontrolname*EndVectorToRaster(TObject *Sender)

(2)Delphi　*imagekitcontrolname*EndVectorToRaster(Sender: TObject)

【引数】

ありません。

【解説】

Vector プロパティの VectorToRaster メソッドを実行した場合に発生します。

GetDragDropFileName（イメージキットコントロール／カスタムイベント）

【機能】
ドラッグ＆ドロップでイメージキットコントロール上にファイルをドロップすると発生します。

【書式】
(1)C++Builder
*imagekitcontrolname***GetDragDropFileName**(TObject *Sender, const UnicodeString FileName, short Page)
(2)Delphi
*imagekitcontrolname***GetDragDropFileName**(Sender: TObject; const FileName: string; Page: Smallint)

【引数】

名称	内容
FileName	イメージキットコントロール上にドロップしたファイル名
Page	マルチイメージファイルのページ番号（0～）

【解説】
エクスプローラやサムネイルコントロールからイメージファイルをイメージキットコントロール上にドラッグ＆ドロップするとイベントが発生します。

【ImageKit7/8/9/10 ActiveX との違い】
・C++Builder で使用する場合、引数の FileName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の FileName が string 型に変更されました。

MagnifierMouseDown（イメージキットコントロール／カスタムイベント）

【機能】
虫眼鏡ウィンドウ表示中にその領域内でマウスのボタンを押した際に発生します。

【書式】
(1)C++Builder
*imagekitcontrolname*MagnifierMouseDown(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y)
(2)Delphi
*imagekitcontrolname*MagnifierMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

【引数】

名称	内容
Button	押されたボタンの種類 mbLeft:　左ボタン mbRight:　右ボタン mbMiddle: 中央ボタン
Shift	〔Alt〕, 〔Ctrl〕, 〔Shift〕の各キーおよびマウスボタンの状態 ssShift:　Shift キーが押されている ssAlt:　Alt キーが押されている ssCtrl:　Ctrl キーが押されている ssLeft:　マウスの左ボタンが押されている ssRight:　マウスの右ボタンが押されている ssMiddle: マウスの中央ボタンが押されている ssDouble: マウスがダブルクリックされた
X, Y	虫眼鏡ウィンドウの中心座標（ピクセル）

【解説】
TMouseButton 型や TShiftState 型については Delphi や C++Builder のヘルプを参照してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の Button が TMouseButton 型に変更されました。
・引数の Shift が TShiftState 型に変更されました。

MagnifierMouseMove（イメージキットコントロール／カスタムイベント）

【機能】
虫眼鏡ウィンドウ表示中にその領域内でマウスを移動した際に発生します。

【書式】
(1)C++Builder　*imagekitcontrolname*MagnifierMouseMove(TObject *Sender, TShiftState Shift, int X, int Y)
(2)Delphi　*imagekitcontrolname*MagnifierMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer)

【引数】

名称	内容
Shift	〔Alt〕,〔Ctrl〕,〔Shift〕の各キーおよびマウスボタンの状態 ssShift:　Shift キーが押されている ssAlt:　Alt キーが押されている ssCtrl:　Ctrl キーが押されている ssLeft:　マウスの左ボタンが押されている ssRight:　マウスの右ボタンが押されている ssMiddle: マウスの中央ボタンが押されている ssDouble: マウスがダブルクリックされた
X, Y	虫眼鏡ウィンドウの中心座標（ピクセル）

【解説】
TShiftState 型については Delphi や C++Builder のヘルプを参照してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の Button が削除されました。
・引数の Shift が TShiftState 型に変更されました。

MagnifierMouseUp(イメージキットコントロール／カスタムイベント)

【機能】
虫眼鏡ウィンドウ表示中にその領域内でマウスのボタンを離した際に発生します。

【書式】
(1)C++Builder
*imagekitcontrolname***MagnifierMouseUp**(TObject *Sender, TMouseButton Button, TShiftState Shift, int X, int Y)
(2)Delphi
*imagekitcontrolname***MagnifierMouseUp**(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer)

【引数】

名称	内容
Button	離されたボタンの種類 mbLeft: 左ボタン mbRight: 右ボタン mbMiddle: 中央ボタン
Shift	〔Alt〕,〔Ctrl〕,〔Shift〕の各キーおよびマウスボタンの状態 ssShift: Shift キーが押されている ssAlt: Alt キーが押されている ssCtrl: Ctrl キーが押されている ssLeft: マウスの左ボタンが押されている ssRight: マウスの右ボタンが押されている ssMiddle: マウスの中央ボタンが押されている ssDouble: マウスがダブルクリックされた
X, Y	虫眼鏡ウィンドウの中心座標(ピクセル)

【解説】
TMouseButton 型や TShiftState 型については Delphi や C++Builder のヘルプを参照してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の Button が TMouseButton 型に変更されました。
・引数の Shift が TShiftState 型に変更されました。

MouseDownImage(イメージキットコントロール／カスタムイベント)

【機能】
マウスのボタンを押した際に発生します。

【書式】
(1)C++Builder
*imagekitcontrolname***MouseDownImage**(TObject *Sender, TMouseButton Button, TShiftState Shift, int OriginX, int OriginY, int ALeft, int ATop, int ARight, int ABottom, double ScaleWidth, double ScaleHeight, int X, int Y)
(2)Delphi
*imagekitcontrolname***MouseDownImage**(Sender: TObject; TMouseButton; Shift: TShiftState; OriginX, OriginY, ALeft, ATop, ARight, ABottom: Integer; ScaleWidth, ScaleHeight: Double; X, Y: Integer)

【引数】

名称	内容
Button	押されたボタンの種類 mbLeft: 左ボタン mbRight: 右ボタン mbMiddle: 中央ボタン
Shift	〔Alt〕,〔Ctrl〕,〔Shift〕の各キーおよびマウスボタンの状態 ssShift: Shift キーが押されている ssAlt: Alt キーが押されている ssCtrl: Ctrl キーが押されている ssLeft: マウスの左ボタンが押されている ssRight: マウスの右ボタンが押されている ssMiddle: マウスの中央ボタンが押されている ssDouble: マウスがダブルクリックされた
OriginX	イメージキットコントロール内でのイメージの表示開始座標(X)
OriginY	イメージキットコントロール内でのイメージの表示開始座標(Y)
ALeft	表示されている実イメージの原点からの位置(左)
ATop	表示されている実イメージの原点からの位置(上)
ARight	表示されている実イメージの原点からの位置(右)
ABottom	表示されている実イメージの原点からの位置(下)
ScaleWidth	実イメージと表示イメージとの横方向のスケール(比率)
ScaleHeight	実イメージと表示イメージとの縦方向のスケール(比率)
X,Y	マウスダウンしたイメージ上の x,y 座標

座標や位置を表す単位はピクセルとなります。

【解説】
マウスのボタンを押した際に発生します。標準の **MouseDown** イベントよりも先に発生します。
ImageHandle や **Layer[Index].ImageHandle** プロパティにイメージが設定されている場合に有効なイベントです。
OriginX,OriginY は **BorderVisible,Appearance,Grad** プロパティの設定値により異なります。
TMouseButton 型や TShiftState 型については Delphi や C++Builder のヘルプを参照してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の Button が TMouseButton 型に変更されました。
・引数の Shift が TShiftState 型に変更されました。

MouseDownSelectLayerImage（イメージキットコントロール／カスタムイベント）

【機能】
階層イメージ上でマウスボタンを押すと発生します。

【書式】
(1)C++Builder　*imagekitcontrolname*MouseDownSelectLayerImage(TObject *Sender, short Index, int X, int Y)
(2)Delphi　*imagekitcontrolname*MouseDownSelectLayerImage(Sender: TObject; Index: Smallint; X, Y: Integer)

【引数】

名称	内容
Index	マウスボタンを押した領域に存在する階層イメージのインデックス（0～99）
X,Y	Index が指す階層イメージの左上からのオフセット（ピクセル）

【解説】
Layer プロパティに実装されている ImageHandle プロパティにイメージが設定され、マウスボタンを押した領域に階層イメージが存在する場合に有効なイベントです。
マウスボタンを押した領域に複数の階層イメージが存在する場合、最前面のイメージが選択されます。
例えば、マウスボタンを押した領域に Layer プロパティのインデックス 0 と 3 のイメージが含まれる場合、3 番が選択されます。
（0 が最背面で 99 が最前面のため。インデックスの小さいものが下側に配置されます。）

MouseMoveImage(イメージキットコントロール／カスタムイベント)

【機能】
マウスを移動させた際に発生します。

【書式】
(1)C++Builder
*imagekitcontrolname***MouseMoveImage**(TObject *Sender, TShiftState Shift, int OriginX, int OriginY, int ALeft, int ATop, int ARight, int ABottom, double ScaleWidth, double ScaleHeight, int X, int Y)
(2)Delphi
*imagekitcontrolname***MouseMoveImage**(Sender: TObject; Shift: TShiftState; OriginX, OriginY, ALeft, ATop, ARight, ABottom: Integer; ScaleWidth, ScaleHeight: Double; X, Y: Integer)

【引数】

名称	内容
Shift	〔Alt〕,〔Ctrl〕,〔Shift〕の各キーおよびマウスボタンの状態 ssShift: Shift キーが押されている ssAlt: Alt キーが押されている ssCtrl: Ctrl キーが押されている ssLeft: マウスの左ボタンが押されている ssRight: マウスの右ボタンが押されている ssMiddle: マウスの中央ボタンが押されている ssDouble: マウスがダブルクリックされた
OriginX	イメージキットコントロール内でのイメージの表示開始座標(X)
OriginY	イメージキットコントロール内でのイメージの表示開始座標(Y)
ALeft	表示されている実イメージの原点からの位置(左)
ATop	表示されている実イメージの原点からの位置(上)
ARight	表示されている実イメージの原点からの位置(右)
ABottom	表示されている実イメージの原点からの位置(下)
ScaleWidth	実イメージと表示イメージとの横方向のスケール(比率)
ScaleHeight	実イメージと表示イメージとの縦方向のスケール(比率)
X,Y	マウスを移動させたイメージ上の x,y 座標

座標や位置を表す単位はピクセルとなります。

【解説】
マウスを移動させた際に発生します。標準の **MouseMove** イベントよりも先に発生します。
ImageHandle や **Layer[Index].ImageHandle** プロパティにイメージが設定されている場合に有効なイベントです。
OriginX,OriginY は **BorderVisible,Appearance,Grad** プロパティの設定値により異なります。
TShiftState 型については Delphi や C++Builder のヘルプを参照してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数に Shift が追加されました。
・引数の Button が削除されました。

MouseUpImage(イメージキットコントロール／カスタムイベント)

【機能】
マウスのボタンを離した際に発生します。

【書式】
(1)C++Builder
*imagekitcontrolname*MouseUpImage(TObject *Sender, TMouseButton Button, TShiftState Shift, int OriginX, int OriginY, int ALeft, int ATop, int ARight, int ABottom, double ScaleWidth, double ScaleHeight, int X, int Y)
(2)Delphi
*imagekitcontrolname*MouseUpImage(Sender: TObject; Button: TMouseButton; Shift: TShiftState; OriginX, OriginY, ALeft, ATop, ARight, ABottom: Integer; ScaleWidth, ScaleHeight: Double; X, Y: Integer)

【引数】

名称	内容
Button	離したボタンの種類 mbLeft: 左ボタン mbRight: 右ボタン mbMiddle: 中央ボタン
Shift	〔Alt〕,〔Ctrl〕,〔Shift〕の各キーおよびマウスボタンの状態 ssShift: Shift キーが押されている ssAlt: Alt キーが押されている ssCtrl: Ctrl キーが押されている ssLeft: マウスの左ボタンが押されている ssRight: マウスの右ボタンが押されている ssMiddle: マウスの中央ボタンが押されている ssDouble: マウスがダブルクリックされた
OriginX	イメージキットコントロール内でのイメージ表示開始座標(X)
OriginY	イメージキットコントロール内でのイメージ表示開始座標(Y)
ALeft	表示されている実イメージの原点からの位置(左)
ATop	表示されている実イメージの原点からの位置(上)
ARight	表示されている実イメージの原点からの位置(右)
ABottom	表示されている実イメージの原点からの位置(下)
ScaleWidth	実イメージと表示イメージとの横方向のスケール(比率)
ScaleHeight	実イメージと表示イメージとの縦方向のスケール(比率)
X,Y	マウスアップしたイメージ上の x,y 座標

座標や位置を表す単位はピクセルとなります。

【解説】
マウスのボタンを離した際に発生します。標準の MouseUp イベントよりも先に発生します。
ImageHandle や Layer[Index].ImageHandle プロパティにイメージが設定されている場合に有効なイベントです。
OriginX,OriginY は BorderVisible,Appearance,Grad プロパティの設定値により異なります。
TMouseButton 型や TShiftState 型については Delphi や C++Builder のヘルプを参照してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の Button が TMouseButton 型に変更されました。
・引数の Shift が TShiftState 型に変更されました。

MouseUpSelectLayerImage(イメージキットコントロール／カスタムイベント)

【機能】
階層イメージ上でマウスボタンを離すと発生します。

【書式】
(1)C++Builder　*imagekitcontrolname***MouseUpSelectLayerImage**(TObject *Sender, short Index, int X, int Y)
(2)Delphi　*imagekitcontrolname***MouseUpSelectLayerImage**(Sender: TObject; Index: Smallint; X, Y: Integer)

【引数】

名称	内容
Index	マウスボタンを離した領域に存在する階層イメージのインデックス(0～99)
X,Y	Index が指す階層イメージの左上からのオフセット(ピクセル)

【解説】
Layer プロパティに実装されている **ImageHandle** プロパティにイメージが設定され、マウスボタンを離した領域に階層イメージが存在する場合に有効なイベントです。
マウスボタンを離した領域に複数の階層イメージが存在する場合、最前面のイメージが選択されます。
例えば、マウスボタンを離した領域に **Layer** プロパティのインデックス 0 と 3 のイメージが含まれる場合、3 番が選択されます。
(0 が最背面で 99 が最前面のため。インデックスの小さいものが下側に配置されます。)

MouseWheelDownImage（イメージキットコントロール／カスタムイベント）

【機能】
マウスホイールを下へ回転すると発生します。

【書式】
(1)C++Builder
*imagekitcontrolname***MouseWheelDownImage**(TObject *Sender, TShiftState Shift, TVIkMouseWheel Direction))
(2)Delphi
*imagekitcontrolname***MouseWheelDownImage**(Sender: TObject; Shift: TShiftState; Direction: TVIkMouseWheel)

【TVIkMouseWheel 型】
ユニット
IkInit
type
TVIkMouseWheel = (vikDisable, vikVertical, vikHorizontal);

【引数】

名称	内容
Shift	〔Alt〕,〔Ctrl〕,〔Shift〕の各キーおよびマウスボタンの状態 ssShift: Shift キーが押されている ssAlt: Alt キーが押されている ssCtrl: Ctrl キーが押されている ssLeft: マウスの左ボタンが押されている ssRight: マウスの右ボタンが押されている ssMiddle: マウスの中央ボタンが押されている ssDouble: マウスがダブルクリックされた
Direction	スクロールする方向（vikDisable：無効, vikVertical:垂直方向, vikHorizontal:水平方向）

【解説】
Direction は **MouseWheel** プロパティで設定した値となります。
TShiftState 型については Delphi や C++Builder のヘルプを参照してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の Shift が TShiftState 型に変更されました。
・列挙型の識別子の先頭に v が付加されました（ActiveX は ikDisable, ikVertical, ikHorizontal）。

MouseWheelUpImage（イメージキットコントロール／カスタムイベント）

【機能】
マウスホイールを上へ回転すると発生します。

【書式】
(1)C++Builder
*imagekitcontrolname***MouseWheelUpImage**(TObject *Sender, TShiftState Shift, TVIkMouseWheel Direction)
(2)Delphi
*imagekitcontrolname***MouseWheelUpImage**(Sender: TObject; Shift: TShiftState; Direction: TVIkMouseWheel)

【TVIkMouseWheel 型】
ユニット
IkInit
type
TVIkMouseWheel = (vikDisable, vikVertical, vikHorizontal);

【引数】

名称	内容
Shift	〔Alt〕,〔Ctrl〕,〔Shift〕の各キーおよびマウスボタンの状態 ssShift: Shift キーが押されている ssAlt: Alt キーが押されている ssCtrl: Ctrl キーが押されている ssLeft: マウスの左ボタンが押されている ssRight: マウスの右ボタンが押されている ssMiddle: マウスの中央ボタンが押されている ssDouble: マウスがダブルクリックされた
Direction	スクロールする方向（vikDisable：無効, vikVertical:垂直方向, vikHorizontal:水平方向）

【解説】
Direction は **MouseWheel** プロパティで設定した値となります。
TShiftState 型については Delphi や C++Builder のヘルプを参照してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の Shift が TShiftState 型に変更されました。
・列挙型の識別子の先頭に v が付加されました（ActiveX は ikDisable, ikVertical, ikHorizontal）。

PanWindow（イメージキットコントロール／カスタムイベント）

【機能】
パンウィンドウを表示もしくは消去すると発生します。

【書式】
(1)C++Builder　*imagekitcontrolname*PanWindow(TObject *Sender, bool Show)
(2)Delphi　*imagekitcontrolname*PanWindow(Sender: TObject; Show: Boolean)

【引数】

名称	内容
Show	表示設定［ True:パンウィンドウを表示、False:パンウィンドウを消去 ］

【解説】
パンウィンドウは PanWindow.Show メソッドを実行することにより表示（消去）できます。

Progress(イメージキットコントロール／カスタムイベント)

【機能】
エフェクト処理およびファイルの読込・保存処理、Vector の RasterToVector メソッド実行中に発生します。

【書式】
(1)C++Builder　*imagekitcontrolname*Progress(TObject *Sender, short Percent)
(2)Delphi　*imagekitcontrolname*Progress(Sender: TObject; Percent: Smallint)

【引数】

名称	内容
Percent	現在処理している%数

【解説】
エフェクト処理では EndDibAccess,StartDibAccess,GetDibPixel,SetDibPixel メソッド以外のメソッド実行時、ファイル処理では FileLoadAsRawData,FileSaveAsRawData,LoadFile(Mem),SaveFile(Mem)メソッド実行時、およびベクトル変換処理(Vector の RasterToVector メソッド)を実行した場合にイベントが発生します。
ただし、その場合でも Caption,Message,ButtonName プロパティを設定して進捗状況のダイアログボックスを表示する場合はイベントは発生いたしません。
処理を中止する場合は、Effect,File,Vector の Cancel プロパティを True に設定します。

Scanning(イメージキットコントロール／カスタムイベント)

【機能】
スキャンデバイスからイメージを取り込んでいる最中に発生します。

【書式】
(1)C++Builder　*imagekitcontrolname*Scanning(TObject *Sender, short Percent)
(2)Delphi　*imagekitcontrolname*Scanning(Sender: TObject; Percent: Smallint)

【引数】

名称	内容
Percent	取り込んでいるイメージの進捗状況(0～100、パーセントで示す)

【解説】
Scanning イベントはメモリ転送のみが対象で、ネイティブ転送やファイル転送では発生いたしません。また、メモリ転送でも圧縮形式の場合と BeforeScan イベントの引数 AWidth と AHeight が-1 の場合はイベントは発生いたしません。
取り込み処理を中止する場合は、イベントの中で Scan.Cancel プロパティを True に設定します。

ScrollDispImage(イメージキットコントロール／カスタムイベント)

【機能】
スクロールバーを使用してスクロールすると発生します。

【書式】
(1)C++Builder
*imagekitcontrolname***ScrollDispImage**(TObject *Sender, TScrollCode ScrollCode, short Pos, TVIkMouseWheel Direction)
(2)Delphi
*imagekitcontrolname***ScrollDispImage**(Sender: TObject; ScrollCode: TScrollCode; Pos: Smallint; Direction: TVIkMouseWheel)

【TVIkMouseWheel 型】
ユニット
IkInit
type
TVIkMouseWheel = (vikDisable, vikVertical, vikHorizontal);

【引数】

名称	内容	
ScrollCode	スクロールバーを操作した方法	
	scLineUp:	上向きまたは左向きのスクロール矢印をクリックした
	scLineDown:	下向きまたは右向きのスクロール矢印をクリックした
	scPageUp:	スクロールボックスの上または左の領域をクリックした
	scPageDown:	スクロールボックスの下または右の領域をクリックした
	scTrack:	スクロールボックスを操作した
	scTop:	スクロールバーの一番上または一番左に移動した
	scBottom:	スクロールバーの一番下または一番右に移動した
Pos	スクロールボックスの現在の位置	
Direction	スクロールする方向(vikVertical:垂直方向, vikHorizontal:水平方向)	

【解説】
実寸表示で **ScrollBar** プロパティを True にした場合に有効なイベントです。
マウスのドラッグによるスクロール(**MouseDragImage** プロパティが True の場合)ではイベントは発生しません。
TScrollCode 型については Delphi や C++Builder のヘルプを参照してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の ScrollCode が TScrollCode 型に変更されました。
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikDisable, ikVertical, ikHorizontal)。

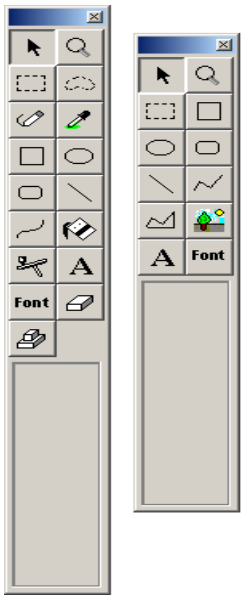
SelEditFunc(イメージキットコントロール／カスタムイベント)

【機能】
イメージ編集ツールバー(ラスタ用およびベクトル用)のボタンを選択すると発生します。

【書式】
(1)C++Builder　*imagekitcontrolname*SelEditFunc(TObject *Sender, TVIkSelEditFunc SelFunction)
(2)Delphi　*imagekitcontrolname*SelEditFunc(Sender: TObject; SelFunction: TVIkSelEditFunc)

【TVIkSelEditFunc 型】
ユニット
IkInit
type
TVIkSelEditFunc = (vikArrow, vikZoom, vikSelect, vikFreeSelect, vikPen, vikSelectColor, vikRectangle, vikEllipse, vikRoundRect, vikLine, vikBezier, vikFillColor, vikAirBrush, vikText, vikFont, vikEraser, vikPolyline, vikPolygon, vikStamp, vikImage);

【引数】

名称	内容	
SelFunction	選択されたボタンの種類 vikArrow:カーソル(ラスタ用、ベクトル用) vikZoom:ズーム(ラスタ用、ベクトル用) vikSelect:選択(ラスタ用、ベクトル用) vikFreeSelect:自由選択(ラスタ用) vikPen:ペン(ラスタ用) vikSelectColor:色選択(ラスタ用) vikRectangle:四角形(ラスタ用、ベクトル用) vikEllipse:楕円(ラスタ用、ベクトル用) vikRoundRect:角丸四角(ラスタ用、ベクトル用) vikLine:直線(ラスタ用、ベクトル用) vikBezier:曲線(ラスタ用) vikFillColor:塗りつぶし(ラスタ用) vikAirBrush:エアブラシ(ラスタ用) vikText:テキスト(ラスタ用、ベクトル用) vikFont:フォント(ラスタ用、ベクトル用) vikEraser:消しゴム(ラスタ用) vikPolyline:連続線(ベクトル用) vikPolygon:多角形(ベクトル用) vikStamp:スタンプ(ラスタ用) vikImage:イメージ(ベクトル用)	イメージ編集ツールバー ラスタ用、ベクトル用 

【解説】
イメージ編集ツールバーは **Edit** プロパティの **ShowToolBar** メソッドを実行することにより表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikArrow, ikZoom, ikSelect, ikFreeSelect, ikPen, ikSelectColor, ikRectangle, ikEllipse, ikRoundRect, ikLine, ikBezier, ikFillColor, ikAirBrush, ikText, ikFont, ikEraser, ikPolyline, ikPolygon, ikStamp, ikImage)。

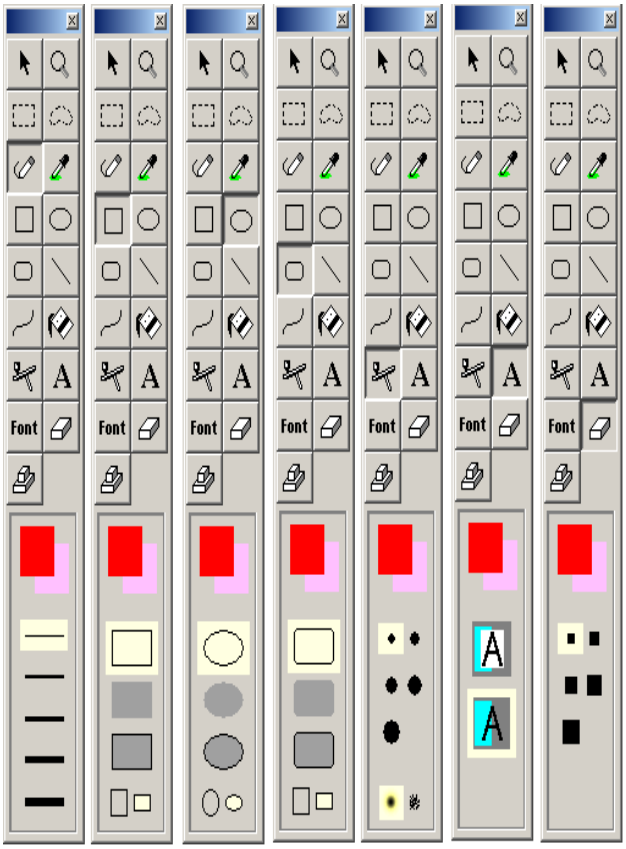
SelEditObj(イメージキットコントロール／カスタムイベント)

【機能】
イメージ編集ツールバー(ラスタ用およびベクトル用)のオブジェクトを選択すると発生します。

【書式】
(1)C++Builder　*imagekitcontrolname***SelEditObj**(TObject *Sender, TVIkSelEditObj SelObject)
(2)Delphi　*imagekitcontrolname***SelEditObj**(Sender: TObject; SelObject: TVIkSelEditObj)

【TVIkSelEditObj 型】
ユニット
IkInit
type
TVIkSelEditObj = (vikFrame, vikFill, vikFillFrame, vikPen1, vikPen2, vikPen3, vikPen4, vikPen5, vikForeColor, vikBackColor, vikTransparent, vikOpaque, vikSquare, vikQuadrangle, vikCircle, vikOval, vikAirBrush1, vikAirBrush2, vikAirBrush3, vikAirBrush4, vikAirBrush5, vikEraser1, vikEraser2, vikEraser3, vikEraser4, vikEraser5, vikBlur, vikSharpness);

【引数】

名称	内容	
SelObject	選択されたオブジェクトの種類 vikFrame:枠付き vikFill:塗り潰し vikFillFrame:枠付きの塗り潰し vikPen1:ペン 1 vikPen2:ペン 2 vikPen3:ペン 3 vikPen4:ペン 4 vikPen5:ペン 5 vikForeColor:前景色 vikBackColor:背景色 vikTransparent:テキストの背景色を透過 vikOpaque:テキストの背景色を設定 vikSquare:正方形 vikQuadrangle:長方形 vikCircle:円 vikOval:楕円 vikAirBrush1:エアブラシ 1 vikAirBrush2:エアブラシ 2 vikAirBrush3:エアブラシ 3 vikAirBrush4:エアブラシ 4 vikAirBrush5:エアブラシ 5 vikEraser1:消しゴム 1 vikEraser2:消しゴム 2 vikEraser3:消しゴム 3 vikEraser4:消しゴム 4 vikEraser5:消しゴム 5 vikBlur:ぼかし有り vikSharpness:ぼかし無し	イメージ編集ツールバー ラスタ用 

【解説】
イメージ編集ツールバーは **Edit** プロパティの **ShowToolBar** メソッドを実行することにより表示されます。
オブジェクトはボタンの下の枠の中に表示されるものです。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikFrame, ikFill, ikFillFrame, ikPen1, ikPen2, ikPen3, ikPen4,

ikPen5, ikForeColor, ikBackColor, ikTransparent, ikOpaque, ikSquare, ikQuadrangle, ikCircle, ikOval, ikAirBrush1, ikAirBrush2, ikAirBrush3, ikAirBrush4, ikAirBrush5, ikEraser1, ikEraser2, ikEraser3, ikEraser4, ikEraser5, ikBlur, ikSharpness)。

StartDispImage(イメージキットコントロール／カスタムイベント)

【機能】
イメージを表示する前に発生します。

【書式】
(1)C++Builder
*imagekitcontrolname*StartDispImage(TObject *Sender, int OriginX, int OriginY, int ALeft, int ATop, int ARight, int ABottom, double ScaleWidth, double ScaleHeight)
(2)Delphi
*imagekitcontrolname*StartDispImage(Sender: TObject; OriginX, OriginY, ALeft, ATop, ARight, ABottom: Integer; ScaleWidth, ScaleHeight: Double)

【引数】

名称	内容
OriginX	イメージキットコントロール内でのイメージの表示開始座標(X)
OriginY	イメージキットコントロール内でのイメージの表示開始座標(Y)
ALeft	表示しようとする実イメージの原点からの位置(左)
ATop	表示しようとする実イメージの原点からの位置(上)
ARight	表示しようとする実イメージの原点からの位置(右)
ABottom	表示しようとする実イメージの原点からの位置(下)
ScaleWidth	実イメージと表示イメージとの横方向のスケール(比率)
ScaleHeight	実イメージと表示イメージとの縦方向のスケール(比率)

座標や位置を表す単位はピクセルとなります。

【解説】
イメージを表示する直前に特定の処理を実行する場合は、イベント内でコードを記述します。
OriginX、OriginY は BorderVisible,Appearance,Grad プロパティの設定値により異なります。

StartPopUpMenu（イメージキットコントロール／カスタムイベント）

【機能】
ポップアップメニューを選択した直後に発生します。

【書式】
(1)C++Builder　*imagekitcontrolname*StartPopUpMenu(TObject *Sender, short MenuNum)
(2)Delphi　*imagekitcontrolname*StartPopUpMenu(Sender: TObject; MenuNum: Smallint)

【引数】

名称	内容
MenuNum	選択されたメニューの種類 **ラスタイメージを編集している場合** 0：全選択 1：切り取り 2：コピー 3：貼り付け 4：回転 5：選択範囲取消し 6：取消し **ベクトルイメージを編集している場合** 0：削除 1：コピー 2：貼り付け 3：回転 4：プロパティ 5：最前面へ移動 6：最背面へ移動 7：取消し

【解説】
ポップアップメニューとは、イメージ編集ツールバーを表示した状態でマウスを右クリックして表示されるメニューリストのことです。

Edit（イメージキットコントロール／カスタム階層プロパティ）

【機能】
イメージを編集するために使用します。

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Action	イメージ編集ツールバーのボタンの種類
AirBrushSize	使用するエアブラシの設定
AirBrushSize1	エアブラシ 1 の半径のサイズ
AirBrushSize2	エアブラシ 2 の半径のサイズ
AirBrushSize3	エアブラシ 3 の半径のサイズ
AirBrushSize4	エアブラシ 4 の半径のサイズ
AirBrushSize5	エアブラシ 5 の半径のサイズ
BackColor	背景色
Blur	エアブラシの種類
Circle	円の種類
ClearText	テキストの初期化
EditEnable	イメージの編集許可設定
EditImageLayerNo	編集対象となるイメージの番号
EditStatus	イメージの編集状態
EnableEditPopupMenu	ポップアップメニューを表示するかどうかの設定
EraserSize	使用する消しゴムの設定
EraserSize1	消しゴム 1 のサイズ
EraserSize2	消しゴム 2 のサイズ
EraserSize3	消しゴム 3 のサイズ
EraserSize4	消しゴム 4 のサイズ
EraserSize5	消しゴム 5 のサイズ
FillPattern	塗り潰し設定
FontBold	フォントの太字設定
FontItalic	フォントの斜体設定
FontName	フォント名の設定
FontSize	フォントのサイズ
FontStrikeOut	テキストの打ち消し線設定
FontTrans	テキストの背景の透過設定
FontUnderline	テキストの下線設定
ForeColor	前景色
Handle	イメージ編集ツールバーのウィンドウハンドル
ImeMode	テキストボックスの IME（かな漢字変換プログラム）の動作
ImeName	キーボード入力をアジア地域の言語の文字に変換するのに使う IME の設定
PenSize	使用するペンの設定
PenSize1	ペン 1 のサイズ
PenSize2	ペン 2 のサイズ
PenSize3	ペン 3 のサイズ
PenSize4	ペン 4 のサイズ
PenSize5	ペン 5 のサイズ
Square	矩形の種類
StampBmpFile	スタンプに使用する BMP ファイル
StampTransBlue	スタンプの透過色の青
StampTransGreen	スタンプの透過色の緑
StampTransRed	スタンプの透過色の赤
Text	描画文字列
ToolBarCaption	イメージ編集ツールバーのキャプションの設定
ToolBarCloseBox	イメージ編集ツールバーの閉じるボタンの設定
ToolBarCursor	イメージ編集ツールバーのマウスカーソルの形状
ToolBarFixed	イメージ編集ツールバーのウィンドウの固定設定

ToolBarLeft	イメージ編集ツールバーを表示する左位置
ToolBarTop	イメージ編集ツールバーを表示する上位置
UndoRasterCountMax	イメージ編集ツールバー(ラスタ)の取り消しの最大回数
UndoVectorCountMax	イメージ編集ツールバー(ベクトル)の取り消しの最大回数

【ImageKit7/8/9/10 ActiveX との違い】
削除されたプロパティ: cCircle

変更されたプロパティ:
Hwnd --> Handle
ToolBarMouseCursorFile --> ToolBarCursor

●メソッド一覧(アルファベット順)

カスタムメソッド	内容
ClearSelect	イメージ編集ツールバーのポップアップメニューの"選択範囲取消し"に相当するメソッド
Copy	イメージ編集ツールバーのポップアップメニューの"コピー"に相当するメソッド
CutAndCopy	イメージ編集ツールバーのポップアップメニューの"切り取り"に相当するメソッド
Delete	イメージ編集ツールバーのポップアップメニューの"削除"に相当するメソッド
GetArrayPointsNum	ポイント数の取得
GetArrayPointsXY	ポイントの座標を取得
Modify	編集した内容をメモリハンドルにコピー
MoveToBack	イメージ編集ツールバーのポップアップメニューの"最背面へ移動"に相当するメソッド
MoveToFront	イメージ編集ツールバーのポップアップメニューの"最前面へ移動"に相当するメソッド
Paste	イメージ編集ツールバーのポップアップメニューの"貼り付け"に相当するメソッド
Rotate	イメージ編集ツールバーのポップアップメニューの"回転"に相当するメソッド
SelectAll	イメージ編集ツールバーのポップアップメニューの"全選択"に相当するメソッド
ShowPropertyDialog	イメージ編集ツールバーのポップアップメニューの"プロパティ"に相当するメソッド
ShowToolBar	イメージ編集ツールバーの表示
Undo	イメージ編集ツールバーのポップアップメニューの"取消し"に相当するメソッド

【ImageKit8/9/10 ActiveX との違い】
変更されたメソッド:
Property --> ShowPropertyDialog

Action(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのボタンの種類を表します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->Action** [= *TVIkSelEditFunc*]
(2)Delphi　*imagekitcontrolname*.**Edit.Action** [= *TVIkSelEditFunc*]

【TVIkSelEditFunc 型】
ユニット
IkInit
type
TVIkSelEditFunc = (vikArrow, vikZoom, vikSelect, vikFreeSelect, vikPen, vikSelectColor, vikRectangle, vikEllipse, vikRoundRect, vikLine, vikBezier, vikFillColor, vikAirBrush, vikText, vikFont, vikEraser, vikPolyline, vikPolygon, vikStamp, vikImage);

【設定値】

値	説明
vikArrow	選択(ラスタ用、ベクトル用)
vikZoom	ズーム(ラスタ用、ベクトル用)
vikSelect	矩形範囲選択(ラスタ用、ベクトル用)
vikFreeSelect	自由範囲選択(ラスタ用)
vikPen	ペン(ラスタ用)
vikSelectColor	色選択(ラスタ用)
vikRectangle	四角形(ラスタ用、ベクトル用)
vikEllipse	楕円(ラスタ用、ベクトル用)
vikRoundRect	角丸四角(ラスタ用、ベクトル用)
vikLine	直線(ラスタ用、ベクトル用)
vikBezier	曲線(ラスタ用)
vikFillColor	塗りつぶし(ラスタ用)
vikAirBrush	エアブラシ(ラスタ用)
vikText	テキスト(ラスタ用、ベクトル用)
vikFont	フォント(ラスタ用、ベクトル用)
vikEraser	消しゴム(ラスタ用)
vikPolyline	連続線(ベクトル用)
vikPolygon	多角形(ベクトル用)
vikStamp	スタンプ(ラスタ用)
vikImage	イメージ(ベクトル用)

【解説】
イメージ編集ツールバーのボタンが押されるとプロパティに値が設定されます。また、プロパティに値を設定することによりイメージ編集ツールバーのボタンを押した状態にすることもできます。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikArrow, ikZoom, ikSelect, ikFreeSelect, ikPen, ikSelectColor, ikRectangle, ikEllipse, ikRoundRect, ikLine, ikBezier, ikFillColor, ikAirBrush, ikText, ikFont, ikEraser, ikPolyline, ikPolygon, ikStamp, ikImage)。

AirBrushSize(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用するエアブラシを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**AirBrushSize** [= *TVIkAirBrushSize*]
(2)Delphi　*imagekitcontrolname*.**Edit.AirBrushSize** [= *TVIkAirBrushSize*]

【TVIkAirBrushSize 型】
ユニット
IkInit
type
TVIkAirBrushSize = (vikAirBrushSize1, vikAirBrushSize2, vikAirBrushSize3, vikAirBrushSize4, vikAirBrushSize5);

【設定値】

値	説明
vikAirBrushSize1	エアブラシ 1(**AirBrushSize1**)
vikAirBrushSize2	エアブラシ 2(**AirBrushSize2**)
vikAirBrushSize3	エアブラシ 3(**AirBrushSize3**)
vikAirBrushSize4	エアブラシ 4(**AirBrushSize4**)
vikAirBrushSize5	エアブラシ 5(**AirBrushSize5**)

【解説】
コードでエアブラシを切り替える場合に使用します。vikAirBrushSize1 の場合は、**AirBrushSize1** プロパティのサイズが有効になります。ただし、イメージ編集ツールバーのオブジェクトを選択した場合は、そちらのサイズが有効になります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikAirBrushSize1, ikAirBrushSize2, ikAirBrushSize3, ikAirBrushSize4, ikAirBrushSize5)。

AirBrushSize1,AirBrushSize2,AirBrushSize3,AirBrushSize4,AirBrushSize5(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用するエアブラシの半径のサイズを取得または設定します。

【書式】
※**AirBrushSize1** にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Edit**->**AirBrushSize1** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Edit.AirBrushSize1** [= *Smallint*]

【設定値】
エアブラシの半径のサイズ。(ピクセル)
デフォルトは **AirBrushSize1** が 4、**AirBrushSize2** が 8、**AirBrushSize3** が 12、**AirBrushSize4** が 16、**AirBrushSize5** が 20 です。

【解説】
5 パターンのエアブラシのサイズを定義できます。
設定したサイズはイメージ編集ツールバーに反映されます。

【値の設定】 実行時

【値の参照】 実行時

BackColor,ForeColor(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用する色情報を取得または設定します。

【書式】
※**BackColor** にて説明(**ForeColor** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Edit**->**BackColor** [= *TColor*]
(2)Delphi　*imagekitcontrolname*.**Edit.BackColor** [= *TColor*]

【設定値】
BackColor は背景色。
ForeColor は前景色。

【解説】
値を設定する場合は、色定数(clRed など)や **RGB**(Red,Green,Blue)の戻り値などを与えます。
TColor 型については Delphi や C++Builder のヘルプを参照してください。

【値の設定】 実行時

【値の参照】 実行時

Blur(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用するエアブラシの種類を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**Blur** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit.Blur** [= *Boolean*]

【設定値】

値	説明
True	ぼかし有り
False	ぼかし無し

【解説】
デフォルトは True です。

【値の設定】 実行時

【値の参照】 実行時

Circle(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用する円の種類を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**Circle** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit**.**Circle** [= *Boolean*]

【設定値】

値	説明
True	円
False	楕円

【解説】
デフォルトは False です。

【値の設定】 実行時

【値の参照】 実行時

ClearText(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのテキストボックスで入力するテキストの初期化を行うかどうかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**ClearText** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit.ClearText** [= *Boolean*]

【設定値】

値	説明
True	初期化する
False	初期化しない

【解説】
デフォルトは False です。
True の場合、テキストボックスを選択して入力するテキストは常に初期化され、False の場合は **Text** プロパティに設定されている文字列が初期表示されます。

【値の設定】 実行時

【値の参照】 不可

EditEnable(イメージキットコントロール／Edit プロパティ)

【機能】
イメージの編集を許可するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**EditEnable** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit**.**EditEnable** [= *Boolean*]

【設定値】

値	説明
True	編集を許可する
False	編集を許可しない

【解説】
デフォルトは False です。True の場合、イメージの編集が可能になります。
ImageHandle プロパティにラスタイメージが設定されている場合は、**ImageHdc** プロパティが有効になります。
ベクトルイメージの場合は、**EditImageLayerNo** プロパティが指すイメージが編集対象となります。
EditEnable プロパティは以前の ImageKit で提供していたディスプレイコントロールの **EditEnable** プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

EditImageLayerNo(イメージキットコントロール／Edit プロパティ)

【機能】
編集対象となるイメージ番号を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**EditImageLayerNo** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Edit.EditImageLayerNo** [= *Smallint*]

【設定値】
-1～99

【解説】
デフォルトは-1 です。
-1 の場合は **ImageHandle** プロパティの指すイメージが編集対象となり、それ以外の場合は **Layer[EditImageLayerNo].ImageHandle** プロパティの指すイメージが編集対象となります。
ただし、0 以上で **Layer[EditImageLayerNo].ImageHandle** プロパティの指すイメージがラスタイメージの場合は編集できません。

【値の設定】 実行時

【値の参照】 実行時

EditStatus(イメージキットコントロール／Edit プロパティ)

【機能】
イメージの編集状態を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**EditStatus** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit.EditStatus** [= *Boolean*]

【設定値】
イメージの編集状態。

値	説明
True	編集済み
False	未編集

【解説】
True の場合は、**Modify** メソッドを実行して編集内容を有効にしてください。
また、アプリケーション終了時にプロパティをチェックして更新することも可能です。
EditStatus プロパティは以前の ImageKit で提供していたディスプレイコントロールの
Edit.RasterEditStatus,**Edit.VectorEditStatus** プロパティに相当します。

【値の設定】 不可

【値の参照】 実行時

EnableEditPopupMenu(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用するポップアップメニューを表示するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->EnableEditPopupMenu** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit.EnableEditPopupMenu** [= *Boolean*]

【設定値】

値	説明
True	ポップアップメニューを表示する
False	ポップアップメニューを表示しない

【解説】
デフォルトは True です。
ポップアップメニューとは、イメージ編集ツールバーを表示した後でマウスを右クリックすると表示されるメニューのことです。

【値の設定】 実行時

【値の参照】 実行時

EraserSize(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用する消しゴムを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**EraserSize** [= *TVIkEraserSize*]
(2)Delphi　*imagekitcontrolname*.**Edit.EraserSize** [= *TVIkEraserSize*]

【TVIkEraserSize 型】
ユニット
IkInit
type
TVIkEraserSize = (vikEraserSize1, vikEraserSize2, vikEraserSize3, vikEraserSize4, vikEraserSize5);

【設定値】

値	説明
vikEraserSize1	消しゴム 1(**EraserSize1**)
vikEraserSize2	消しゴム 2(**EraserSize2**)
vikEraserSize3	消しゴム 3(**EraserSize3**)
vikEraserSize4	消しゴム 4(**EraserSize4**)
vikEraserSize5	消しゴム 5(**EraserSize5**)

【解説】
コードで消しゴムを切り替える場合に使用します。vikEraserSize1 の場合は、**EraserSize1** プロパティのサイズが有効になります。ただし、イメージ編集ツールバーのオブジェクトを選択した場合は、そちらのサイズが有効になります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikEraserSize1, ikEraserSize2, ikEraserSize3, ikEraserSize4, ikEraserSize5)。

EraserSize1,EraserSize2,EraserSize3,EraserSize4,EraserSize5(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用する消しゴムのサイズを取得または設定します。

【書式】
※**EraserSize1** にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Edit**->**EraserSize1** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Edit.EraserSize1** [= *Smallint*]

【設定値】
消しゴムの太さ。(ピクセル)
デフォルトは **EraserSize1** が 1、**EraserSize2** が 3、**EraserSize3** が 6、**EraserSize4** が 9、**EraserSize5** が 12 です。

【解説】
5 パターンの消しゴムの太さを定義できます。
設定した太さはイメージ編集ツールバーに反映されます。

【値の設定】 実行時

【値の参照】 実行時

FillPattern(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用する塗り潰しパターンを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->FillPattern** [= *TVIkFillPattern*]
(2)Delphi　*imagekitcontrolname*.**Edit.FillPattern** [= *TVIkFillPattern*]

【TVIkFillPattern 型】
ユニット
IkInit
type
TVIkFillPattern = (vikFramePattern, vikFillPattern, vikFillFramePattern);

【設定値】

値	説明
vikFramePattern	枠付き
vikFillPattern	塗り潰し
vikFillFramePattern	枠付きの塗り潰し

【解説】
四角、楕円、角丸四角を描画する際の塗り潰しパターンです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikFramePattern, ikFillPattern, ikFillFramePattern)。

FontBold,FontItalic ,FontStrikeOut,FontTrans,FontUnderline (イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのフォントのスタイル・下線・打ち消し線などを取得または設定します。

【書式】
※**FontBold** にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Edit**->**FontBold** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit.FontBold** [= *Boolean*]

【設定値】
FontBold はフォントを太字にするかしないかの設定。デフォルトは False。
FontItalic はフォントを斜体にするかしないかの設定。デフォルトは False。
FontStrikeOut はテキストに打ち消し線を付加するかしないかの設定。デフォルトは False。
FontTrans はテキストの背景色を透過にするかしないかの設定。デフォルトは True。
FontUnderline はテキストに下線を付加するかしないかの設定。デフォルトは False。

値	説明
True	する
False	しない

【解説】
イメージ編集ツールバーを使用してテキストを描画する際の情報です。

【値の設定】 実行時

【値の参照】 実行時

FontName,Text(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーを使用してテキストを描画する際のフォント名や文字列を取得または設定します。

【書式】
※**FontName** にて説明(**Text** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Edit**->**FontName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Edit.FontName** [= *string*]

【設定値】
FontName はフォントの名称(31 文字以内)。
Text は描画する文字列。

【解説】
イメージ編集ツールバーを使用してテキストを描画する際に有効です。

【値の設定】 実行時

【値の参照】 実行時

FontSize(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーを使用してテキストを描画する際のフォントのサイズを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->FontSize** [*= short*]
(2)Delphi　*imagekitcontrolname*.**Edit.FontSize** [*= Smallint*]

【設定値】
フォントのサイズ(ポイント)

【解説】
イメージ編集ツールバーを使用してテキストを描画する際に有効です。

【値の設定】 実行時

【値の参照】 実行時

Handle(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのウィンドウハンドルを取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**Handle** [= *HWND*]
(2)Delphi　*imagekitcontrolname*.**Edit.Handle** [= *HWND*]

【参照値】
イメージ編集ツールバーのウィンドウハンドル。

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
プロパティの名称が **Hwnd** から変更されました。

ImeMode(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのテキストボックスで使用する IME(かな漢字変換プログラム)の動作を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->ImeMode** [= *TImeMode*]
(2)Delphi　*imagekitcontrolname*.**Edit.ImeMode** [= *TImeMode*]

【設定値】

値	説明
imDisable	IME を終了する。imDisable は簡体字中国語，繁体字中国語，韓国語 IME では無効
imClose	IME 変換ウィンドウを閉じるが、IME はバックグラウンドで動作する IME はホットキーによって再びアクティブ化できる
imOpen	IME 変換ウィンドウを開く。この変換モードはほとんど使用されない
imDontCare	使用できない場合でも IME を起動する。この変換モードはほとんど使用されない
imSAlpha	IME 変換ウィンドウを開いて、半角ローマ字入力を受け付けるように変換モードを設定する
imAlpha	IME 変換ウィンドウを開いて、全角ローマ字入力を受け付けるように変換モードを設定する
imHira	IME 変換ウィンドウを開いて、全角ひらがな入力を受け付けるように変換モードを設定する imHira は日本語 IME でのみ使用できる
imSKata	IME 変換ウィンドウを開いて，半角カタカナ入力を受け付けるように変換モードを設定する imSKata は日本語 IME でのみ使用できる
imKata	IME 変換ウィンドウを開いて，全角カタカナ入力を受け付けるように変換モードを設定する imKata は日本語 IME でのみ使用できる
imChinese	IME 変換ウィンドウを開いて、2 バイト中国語入力を受け付けるように変換モードを設定する imChinese は中国語 IME でのみ使用できる
imSHanguel	IME 変換ウィンドウを開いて、1 バイト韓国語入力を受け付けるように変換モードを設定する imSHanguel は韓国語 IME でのみ使用できる
imHanguel	IME 変換ウィンドウを開いて、2 バイト韓国語入力を受け付けるように変換モードを設定する imHanguel は韓国語 IME でのみ使用できる

【解説】
デフォルトは imDontCare です。
TImeMode 型については Delphi や C++Builder のヘルプを参照してください。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
整数型から列挙型に変更されました。

ImeName(イメージキットコントロール／Edit プロパティ)

【機能】
キーボード入力をアジア地域の言語の文字に変換するのに使う IME(かな漢字変換プログラム)を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**ImeName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Edit.ImeName** [= *string*]

【設定値】
IME の名称

【解説】
イメージ編集ツールバーのテキストボックスに入力する際に使用する IME です。デフォルトは空文字列です。
Windows コントロールパネルでインストールされている IME のいずれかを設定します。現在インストールされている IME のリストは Screen 変数(C++Builder/Delphi の Forms ユニット[XE2 以降では先頭にユニットスコープ名が付加され、Vcl.Forms])から取得できます。
プロパティに使用できない IME を設定すると、アプリケーションの起動時にアクティブだった IME が使用され、例外は生成されません。

【値の設定】 実行時

【値の参照】 実行時

PenSize(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用するペンを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**PenSize** [= *TVIkPenSize*]
(2)Delphi　*imagekitcontrolname*.**Edit.PenSize** [= *TVIkPenSize*]

【TVIkPenSize 型】
ユニット
IkInit
type
TVIkPenSize = (vikPenSize1, vikPenSize2, vikPenSize3, vikPenSize4, vikPenSize5);

【設定値】

値	説明
vikPenSize1	ペン 1(**PenSize1**)
vikPenSize2	ペン 2(**PenSize2**)
vikPenSize3	ペン 3(**PenSize3**)
vikPenSize4	ペン 4(**PenSize4**)
vikPenSize5	ペン 5(**PenSize5**)

【解説】
コードでペンを切り替える場合に使用します。vikPenSize1 の場合は、**PenSize1** プロパティのサイズが有効になります。
ただし、イメージ編集ツールバーのオブジェクトを選択した場合は、そちらのサイズが有効になります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikPenSize1, ikPenSize2, ikPenSize3, ikPenSize4, ikPenSize5)。

PenSize1,PenSize2,PenSize3,PenSize4,PenSize5(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用するペンのサイズを取得または設定します。

【書式】
※**PenSize1** にて説明(その他も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->**Edit**->**PenSize1** [= *short*]
(2)Delphi　　*imagekitcontrolname*.**Edit.PenSize1** [= *Smallint*]

【設定値】
ペンの太さ。(ピクセル)
デフォルトは **PenSize1** が 1、**PenSize2** が 2、**PenSize3** が 3、**PenSize4** が 4、**PenSize5** が 5 です。

【解説】
5 パターンのペンの太さを定義できます。
設定した太さはイメージ編集ツールバーに反映されます。

【値の設定】 実行時

【値の参照】 実行時

Square(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーで使用する矩形の種類を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**Square** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit.Square** [= *Boolean*]

【設定値】

値	説明
True	正方形
False	長方形

【解説】
デフォルトは False です。

【値の設定】 実行時

【値の参照】 実行時

StampBmpFile(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのスタンプで使用するビットマップファイルを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->StampBmpFile** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Edit.StampBmpFile** [= *string*]

【設定値】
スタンプに用いるビットマップファイル名。

【解説】
スタンプの透過色を設定する場合は、ファイル名よりも先に透過色を設定する必要があります。

透過色を白にする例:
(1)C++Builder
```
VImageKit1->Edit->StampTransBlue = 255;
VImageKit1->Edit->StampTransGreen = 255;
VImageKit1->Edit->StampTransRed = 255;
VImageKit1->Edit->StampBmpFileName = "Stamp.bmp";
```
(2)Delphi
```
VImageKit1.Edit.StampTransBlue := 255;
VImageKit1.Edit.StampTransGreen := 255;
VImageKit1.Edit.StampTransRed := 255;
VImageKit1.Edit.StampBmpFileName := 'Stamp.bmp';
```

【値の設定】 実行時

【値の参照】 実行時

StampTransBlue,StampTransGreen,StampTransRed(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのスタンプの透過色を取得または設定します。

【書式】
※**StampTransBlue** にて説明(**StampTransGreen**、**StampTransRed** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Edit**->**StampTransBlue** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Edit.StampTransBlue** [= *Smallint*]

【設定値】
StampTransBlue はスタンプの透過色の青。
StampTransGreen はスタンプの透過色の緑。
StampTransRed はスタンプの透過色の赤。

0～255。

【解説】
編集イメージが白黒 2 値(1 ビット)の場合は、**StampTransBlue**,**StampTransGreen**,**StampTransRed** プロパティに 0 を設定してください。

スタンプの透過色を設定する場合は、ファイル名よりも先に透過色を設定する必要があります。

透過色を白にする例(Delphi):

```
VImageKit1.Edit.StampTransBlue := 255;
VImageKit1.Edit.StampTransGreen := 255;
VImageKit1.Edit.StampTransRed := 255;
VImageKit1.Edit.StampBmpFileName := 'Stamp.bmp';
```

【値の設定】 実行時

【値の参照】 実行時

ToolBarCaption(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのキャプションを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**ToolBarCaption** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Edit.ToolBarCaption** [= *string*]

【設定値】
イメージ編集ツールバーのキャプションに表示する文字列。

【解説】
ToolBarCaption プロパティに空の文字列に設定し **ToolBarCloseBox** プロパティを False にすると、タイトルバーが表示されなくなり、マウスでイメージ編集ツールバーの移動ができなくなります。

【値の設定】 実行時

【値の参照】 実行時

ToolBarCloseBox(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーに閉じるボタンを付加するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->ToolBarCloseBox** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit.ToolBarCloseBox** [= *Boolean*]

【設定値】

値	説明
True	閉じるボタン(×ボタン)を付加する
False	閉じるボタン(×ボタン)を付加しない

【値の設定】 実行時

【値の参照】 実行時

ToolBarCursor(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのマウスカーソルの形状を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->ToolBarCursor** [= *TCursor*]
(2)Delphi　*imagekitcontrolname*.**Edit.ToolBarCursor** [= *TCursor*]

【設定値】
crDefault や crArrow など。詳しくは Delphi や C++Builder のヘルプを参照のこと。

【解説】
デフォルトは crArrow です。
設定したカーソル形状が有効になるのは、イメージ編集ツールバー内にマウスカーソルが入った場合です。
カスタムカーソルを割り当てる場合は、Delphi や C++Builder のヘルプの TControl.Cursor の例を参照してください。
ToolBarCursor プロパティは以前の ImageKit で提供していたディスプレイコントロールの **Edit.ToolBarCursor** プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
・プロパティの名称が **ToolBarMouseCursorFile** から変更されました。
・文字列型から TCursor 型に変更されました。

ToolBarFixed(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーのウィンドウを固定するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->ToolBarFixed** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Edit.ToolBarFixed** [= *Boolean*]

【設定値】

値	説明
True	ウィンドウは移動可能
False	ウィンドウは移動不可

【値の設定】 実行時

【値の参照】 実行時

ToolBarLeft,ToolBarTop(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーを表示する左上位置をピクセル単位で示します。

【書式】
※**ToolBarLeft** にて説明(**ToolBarTop** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Edit**->**ToolBarLeft** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Edit.ToolBarLeft** [= *Smallint*]

【設定値】
スクリーン左上からのイメージ編集ツールバーの左上位置(ピクセル単位)。
ToolBarLeft は、左位置
ToolBarTop は、上位置

【値の設定】 実行時

【値の参照】 実行時

UndoRasterCountMax,UndoVectorCountMax(イメージキットコントロール／Edit プロパティ)

【機能】
イメージ編集ツールバーの取り消しの最大回数を取得または設定します。

【書式】
※**UndoRasterCountMax** にて説明(**UndoVectorCountMax** も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->**Edit**->**UndoRasterCountMax** [= *short*]
(2)Delphi　　*imagekitcontrolname*.**Edit.UndoRasterCountMax** [= *Smallint*]

【設定値】
取り消しの最大回数(デフォルトは 50)。有効範囲は 0 から 50 まで。
UndoRasterCountMax はラスタイメージ用の取り消し最大回数。
UndoVectorCountMax はベクトルイメージ用の取り消し最大回数。

【解説】
取り消し回数を大きくすると便利な反面、メモリを多く消費します。
0 を設定した場合、イメージ編集ツールバーのポップアップメニューで「取消し」を実行することはできません。
取り消し回数はイメージ編集ツールバーを表示する前に設定してください。(イメージ編集ツールバー表示中は変更不可)

例(Delphi):

```
(1)ラスタイメージ用
VImageKit1.Edit.ToolBarTop := 50;
VImageKit1.Edit.ToolBarLeft := 100;
VImageKit1.Edit.ToolBarCloseBox := True;
VImageKit1.Edit.ToolBarCaption := 'ToolBar';
VImageKit1.Edit.FontName := 'Arial';
VImageKit1.Edit.FontSize := 12;
VImageKit1.Edit.StampBmpFile := CurrentDir + 'Stamp.bmp';
VImageKit1.Edit.UndoRasterCountMax := 10;

VImageKit1.Edit.EditEnable := True;
VImageKit1.Edit.EditImageLayerNo := -1;
VImageKit1.Edit.ShowToolbar(vikRaster);

(2)ベクトルイメージ用
VImageKit1.Edit.ToolBarTop := 50;
VImageKit1.Edit.ToolBarLeft := 100;
VImageKit1.Edit.ToolBarCloseBox := True;
VImageKit1.Edit.ToolBarCaption := 'ToolBar';
VImageKit1.Edit.FontName := 'Arial';
VImageKit1.Edit.FontSize := 12;
VImageKit1.Edit.UndoVectorCountMax := 10;

VImageKit1.Edit.EditEnable := True;
VImageKit1.Edit.EditImageLayerNo := 2;
VImageKit1.Edit.ShowToolbar(vikVector);
```

【値の設定】 実行時

【値の参照】 実行時

ClearSelect（イメージキットコントロール／Edit メソッド）

【機能】
選択して決定した範囲を取消し、解除します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**ClearSelect**()
(2)Delphi　*imagekitcontrolname*.**Edit.ClearSelect**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー（ラスタ）のポップアップメニューの"選択範囲取消し"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

Copy(イメージキットコントロール／Edit メソッド)

【機能】
イメージ編集ツールバー(ラスタ)の場合:
選択範囲のイメージをクリップボードにコピーします。
イメージ編集ツールバー(ベクトル)の場合:
選択した要素をコピーします。

【書式】
(1)C++Builder *imagekitcontrolname*->**Edit**->**Copy**()
(2)Delphi *imagekitcontrolname*.**Edit.Copy**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー(ラスタ、ベクトル)のポップアップメニューの"コピー"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

CutAndCopy（イメージキットコントロール／Edit メソッド）

【機能】
選択範囲を切り取り、切り取ったイメージをクリップボードにコピーします。切り取りで、空いた領域は自動的にバックカラー（背景色）で埋められます。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->CutAndCopy**()
(2)Delphi　*imagekitcontrolname*.**Edit.CutAndCopy**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー（ラスタ）のポップアップメニューの"切り取り"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

Delete（イメージキットコントロール／Edit メソッド）

【機能】
選択した要素を削除します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**Delete**()
(2)Delphi　*imagekitcontrolname*.**Edit.Delete**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー（ベクトル）のポップアップメニューの"削除"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

GetArrayPointsNum（イメージキットコントロール／Edit メソッド）

【機能】
イメージ編集ツールバーの範囲選択と自由範囲選択で選択されたポイントの数を取得します。

【書式】
(1)C++Builder　[*int* =]*imagekitcontrolname*->**Edit->GetArrayPointsNum**()
(2)Delphi　[*Integer* =]*imagekitcontrolname*.**Edit.GetArrayPointsNum**

【引数】
ありません。

【戻り値】
ポイントの数（失敗した場合は 0）。

【解説】
選択されたポイントの座標値を取得するには、当メソッドの戻り値の数を要素数とする TPoint 型の配列を作成し、**GetArrayPointsXY** メソッドを実行します。

GetArrayPointsXY（イメージキットコントロール／Edit メソッド）

【機能】
イメージ編集ツールバーの範囲選択と自由範囲選択で選択されたポイントの座標値を取得します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Edit**->**GetArrayPointsXY**(TPoint * Points, const int Points_Size)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Edit.GetArrayPointsXY**(var Points: array of TPoint)

【引数】

名称	内容
Points	ポイントの x,y 座標を取得する配列

※C++Builder の場合、配列の要素数-1 を Points_Size に与えます。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
選択されたポイントの座標値を取得するには、**GetArrayPointsNum** メソッドの戻り値の数を要素数とする TPoint 型の配列を作成し、当メソッドを実行します。取得される座標はピクセル単位です。

選択されたポイントの座標値を取得するコード例：
(1)C++Builder

```
TPoint *Points;
bool ret
int count;

count = VImageKit1->Edit->GetArrayPointsNum();
if (count == 0) return;
Points = new TPoint[count];
ret = VImageKit1->Edit->GetArrayPointsXY(Points, count - 1);
// 様々な処理
delete[] Points;
```

(2)Delphi

```
Points: array of TPoint;
ret: Boolean;
count: Integer;

count := VImageKit1.Edit.GetArrayPointsNum;
if count = 0 then Exit;
SetLength(Points, count);
ret := VImageKit1.Edit.GetArrayPointsXY(Points);
```

【ImageKit7/8/9/10 ActiveX との違い】
引数の x,y が TPoint 型に変更されました。

Modify（イメージキットコントロール／Edit メソッド）

【機能】

ImageHdc プロパティで編集したイメージをメモリハンドルへコピーします。

【書式】

(1)C++Builder　[*bool =*]*imagekitcontrolname*->Edit->Modify()

(2)Delphi　[*Boolean =*]*imagekitcontrolname*.Edit.Modify

【引数】

ありません。

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

編集対象がラスタイメージの場合は、ImageHdc プロパティに描画されている内容を ImageHandle プロパティが示すメモリハンドルにコピーします。

編集対象がベクトルイメージの場合は、Modify メソッドとは関係なくイメージ編集ツールバーの操作毎に ImageHandle プロパティおよび Layer(EditImageLayerNo).ImageHandle プロパティが示すメモリハンドルが更新されます（EditImageLayerNo プロパティの値による）。

MoveToBack(イメージキットコントロール／Edit メソッド)

【機能】
選択した要素を最背面へ移動します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->MoveToBack**()
(2)Delphi　*imagekitcontrolname*.**Edit.MoveToBack**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー(ベクトル)のポップアップメニューの"最背面へ移動"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

MoveToFront（イメージキットコントロール／Edit メソッド）

【機能】
選択した要素を最前面へ移動します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**MoveToFront**()
(2)Delphi　*imagekitcontrolname*.**Edit.MoveToFront**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー（ベクトル）のポップアップメニューの"最前面へ移動"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

Paste(イメージキットコントロール／Edit メソッド)

【機能】

イメージ編集ツールバー(ラスタ)の場合:

クリップボードにあるイメージを現在のイメージ上に貼り付けます。貼り付けた直後に貼り付け位置をドラッグで決定します。

イメージ編集ツールバー(ベクトル)の場合:

コピーした選択した要素を貼り付けます。

【書式】

(1)C++Builder　*imagekitcontrolname*->**Edit**->**Paste**()

(2)Delphi　*imagekitcontrolname*.**Edit.Paste**

【引数】

ありません。

【戻り値】

ありません。

【解説】

イメージ編集ツールバー(ラスタ、ベクトル)のポップアップメニューの"貼り付け"に相当するメソッドです。

マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

Rotate(イメージキットコントロール／Edit メソッド)

【機能】
イメージ編集ツールバー(ラスタ)の場合:
選択範囲のイメージを回転させます。
イメージ編集ツールバー(ベクトル)の場合:
選択した要素を回転させます。

【書式】
(1)C++Builder *imagekitcontrolname*->**Edit**->**Rotate**()
(2)Delphi *imagekitcontrolname*.**Edit.Rotate**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー(ラスタ、ベクトル)のポップアップメニューの"回転"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

SelectAll(イメージキットコントロール／Edit メソッド)

【機能】
部分的な領域ではなく、イメージ全体を選択範囲とします。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->SelectAll**()
(2)Delphi　*imagekitcontrolname*.**Edit.SelectAll**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー(ラスタ)のポップアップメニューの"全選択"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

ShowPropertyDialog（イメージキットコントロール／Edit メソッド）

【機能】
選択した要素のプロパティを確認・変更できます。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit->ShowPropertyDialog**()
(2)Delphi　*imagekitcontrolname*.**Edit.ShowPropertyDialog**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー（ベクトル）のポップアップメニューの"プロパティ"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

【ImageKit8/9/10 ActiveX との違い】
メソッドの名称が **Property** から変更されました。

ShowToolBar(イメージキットコントロール／Edit メソッド)

【機能】
ToolBarLeft、ToolBarTop プロパティで設定した位置にイメージ編集ツールバーを表示します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Edit**->**ShowToolBar**(TVIkEditToolBarMode Mode)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Edit.ShowToolBar**(Mode: TVIkEditToolBarMode)

【TVIkEditToolBarMode 型】
ユニット
IkInit
type
TVIkEditToolBarMode = (vikHide, vikRaster, vikVector);

【引数】

名称	内容
Mode	表示モード(vikHide:非表示、vikRaster:ラスタイメージ用、vikVector:ベクトルイメージ用)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ToolBarLeft、ToolBarTop プロパティに適切な値が設定されている必要があります。

編集対象となるイメージは ImageHandle プロパティと Layer[EditImageLayerNo].ImageHandle プロパティです(EditImageLayerNo プロパティの値による)。
編集対象イメージは Modify メソッドを行うことによりメモリハンドルに反映されます。

イメージキットコントロールを配置したフォームをモーダルで表示する場合は、フォームを表示した後でイメージ編集ツールバーを表示するようにしてください。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は vikHide, vikRaster, vikVector)。

Undo(イメージキットコントロール／Edit メソッド)

【機能】
一つ前に行った操作を取消し、元の状態に戻します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Edit**->**Undo**()
(2)Delphi　*imagekitcontrolname*.**Edit.Undo**

【引数】
ありません。

【戻り値】
ありません。

【解説】
イメージ編集ツールバー(ラスタ、ベクトル)のポップアップメニューの"取消し"に相当するメソッドです。
マウスの右クリックでポップアップメニューを表示せず、プログラムで操作する場合に使用します。

Effect(イメージキットコントロール／カスタム階層プロパティ)

【機能】
エフェクト処理を実現します。

●RGB と YCrCb カラースペースについて
エフェクト処理ではしばし、RGB と YCrCb の変換が内部的に行われます。
次に計算式を示します。
なお、Y は輝度情報、CrCb は色相情報を表します。

(1)RGB から YCrCb へ
Y = (0.29900 * R) + (0.58700 * G) + (0.11400 * B)
Cr = (0.50000 * R) + (-0.41869 * G) + (-0.08131 * B) + 128
Cb = (-0.16874 * R) + (-0.33126 * G) + (0.50000 * B) + 128

(2)YCrCb から RGB へ
R = Y + (1.402 * (Cr - 128))
G = Y - (0.34414 * (Cb - 128)) - (0.71414 * (Cr - 128))
B = Y + (1.772 * (Cb - 128))

●プロパティ一覧(アルファベット順)

カスタムプロパティ	内容
ButtonName	処理中のダイアログボックスに表示するボタンの名称を設定
Cancel	処理の中止の設定
Caption	処理中のダイアログボックスに表示するタイトルバーの名称を設定
InOut	選択範囲の内外の設定
LayerAlphaChannel	Layer メソッドでアルファチャンネルを考慮するかどうかを設定
MaskFileName	入力用のマスクイメージのファイル名
MaskImageHandle	入出力用のマスクイメージのメモリハンドル
Message	処理中のダイアログボックスに表示するメッセージを設定
RectBottom	四角形の下位置を設定
RectLeft	四角形の左位置を設定
RectRight	四角形の右位置を設定
RectTop	四角形の上位置を設定
SelectMode	選択範囲の種類を設定

●メソッド一覧(アルファベット順)

カスタムメソッド	内容
Affine	アフィン変換処理
AntiAlias	アンチエリアシング処理
AutoSelectImage	自動選択処理
Blur	ぼかし処理
Canvas	キャンバス地効果
CheckSecretImage	透かしの取り出し処理
Chroma	彩度補正処理
ConvertColor	減色・増色処理
CustomFilter	カスタムフィルタ
CutRectImage	矩形切り出し処理
Emboss	エンボス処理
EndDibAccess	DIB アクセスの終了処理
GetDibPixel	DIB からピクセル値を取得
GlassTile	ガラスタイル効果
LayerImage	ラスタイメージの重ね合わせ処理
Lens	レンズ効果
MakeRGBAImage	RGB と A から RGBA の 32 ビットイメージを作成

Mosaic	モザイク処理
MotionBlur	モーションぼかし処理
OilPaint	油絵風効果
Outline	輪郭抽出処理
Panorama	パノラマ処理
PasteImage	ラスタイメージの貼り付け処理
RedEyeRemoval	赤目補正
RemoveNoise	ノイズ除去処理
Resize	拡大縮小処理
RGBGamma	RGB 値のガンマ補正処理
RGBLevel	RGB 値の加減処理
RGBRev	RGB 値の反転処理
RGBSpline	RGB 値のスプライン補正処理
Ripple	波紋効果
Rotation	回転処理
SelectImage	自由選択処理
SetDibPixel	DIB へピクセル値を設定
SetGray	カラーイメージをグレースケールへ変換
SetSecretImage	透かしの埋め込み処理
Sharp	シャープネス処理
SplitRGBAImage	RGBA の 32 ビットイメージを RGB(24 ビット)と A(8 ビット)に分割
StartDibAccess	DIB アクセスの開始処理
UnifyColor	色のばらつきを修正
Waves	さざ波効果
WhirlPinch	ねじりつまみ効果
YCCGamma	YCrCb 値のガンマ補正処理
YCCLevel	YCrCb 値の加減処理
YCCRev	YCrCb 値の反転処理
YCCSpline	YCrCb 値のスプライン補正処理

ButtonName,Caption,Message(イメージキットコントロール／Effect プロパティ)

【機能】
エフェクト処理中のダイアログボックスに表示するボタン、キャプション、メッセージを取得または設定します。

【書式】
※**ButtonName** にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Effect**->**ButtonName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Effect**.**ButtonName** [= *string*]

【設定値】
ButtonName プロパティは、処理中のダイアログボックスのボタンに表示する文字列。
Caption プロパティは、処理中のダイアログボックスのタイトルバーに表示する文字列。
Message プロパティは、処理中のダイアログボックスの中央に表示する文字列。

【解説】
ButtonName,**Caption**,**Message** プロパティ全てに何も設定しなかった場合は、進捗状況のダイアログボックスは表示されず、**Progress** イベントが発生します。
逆にどれか一つでも有効な文字列を設定した場合は、進捗状況ダイアログボックスが表示され、**Progress** イベントは発生しません。

【値の設定】 実行時

【値の参照】 実行時

Cancel(イメージキットコントロール／Effect プロパティ)

【機能】
エフェクト処理を中止するかどうかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Effect**->**Cancel** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Effect**.**Cancel** [= *Boolean*]

【設定値】

値	説明
True	処理を中止する
False	処理を中止しない

【解説】
Cancel プロパティに True を設定すると、エフェクト処理を中止できます。

【値の設定】 実行時

【値の参照】 不可

InOut(イメージキットコントロール／Effect プロパティ)

【機能】
エフェクト処理の選択範囲の内外を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Effect**->**InOut** [= *TVIkInOutSelectMode*]
(2)Delphi　*imagekitcontrolname*.**Effect**.**InOut** [= *TVIkInOutSelectMode*]

【TVIkInOutSelectMode 型】
```
ユニット
  IkInit
type
  TVIkInOutSelectMode = (vikOutside, vikInside);
```

【設定値】

値	説明
vikOutside	選択範囲の外側(境界含まない)
vikInside	選択範囲の内側(境界含む)

【解説】
設定したプロパティは、**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合に有効になります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikOutside, ikInside)。

LayerAlphaChannel(イメージキットコントロール／Effect プロパティ)

【機能】

Layer メソッドでアルファチャンネルを考慮するかどうかを取得または設定します。

【書式】

(1)C++Builder　*imagekitcontrolname*->**Effect**->**LayerAlphaChannel** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Effect**.**LayerAlphaChannel** [= *Boolean*]

【設定値】

値	説明
True	アルファチャンネルを考慮する
False	アルファチャンネルを考慮しない(デフォルト)

【解説】

LayerAlphaChannel プロパティに True を設定する例として、例えば JPEG 画像の上にアルファチャンネルの設定のある PNG 画像を重ね合わせることが考えられます。

【値の設定】 実行時

【値の参照】 実行時

MaskFileName(イメージキットコントロール／Effect プロパティ)

【機能】
入力用のマスクイメージのファイル名を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Effect**->**MaskFileName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Effect**.**MaskFileName** [= *string*]

【設定値】
マスクイメージのファイル名

【解説】
処理対象となるマスクイメージのファイル名を設定します。マスクイメージは **ImageHandle** プロパティに対するイメージです。
マスクイメージは 1 ビット(白黒 2 値)イメージでなければなりません。
MaskImageHandle プロパティと **MaskFileName** プロパティは両方ともマスクイメージを表しますが、最後に設定されたプロパティがエフェクト処理に使用されるマスクイメージとなります。
エフェクト処理が成功すると **MaskImageHandle** プロパティに処理後のマスクイメージが設定されます。

【値の設定】 実行時

【値の参照】 実行時

MaskImageHandle(イメージキットコントロール／Effect プロパティ)

【機能】
入出力用のマスクイメージのメモリハンドルを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Effect**->**MaskImageHandle** [= *NativeUInt*]
(2)Delphi　*imagekitcontrolname*.**Effect**.**MaskImageHandle** [= *THandle*]

【設定値】
マスクイメージのメモリハンドル

【解説】
処理対象となるマスクイメージのメモリハンドルを設定します。マスクイメージは **ImageHandle** プロパティに対するイメージです。
マスクイメージは 1 ビット(白黒 2 値)イメージでなければなりません。
MaskImageHandle プロパティと **MaskFileName** プロパティは両方ともマスクイメージを表しますが、最後に設定されたプロパティがエフェクト処理に使用されるマスクイメージとなります。
エフェクト処理が成功すると **MaskImageHandle** プロパティは自動的に更新されます。
MaskImageHandle プロパティは内部で更新されますので、明示的に解放する必要はありません。

MaskImageHandle プロパティは以前の ImageKit で提供していたエフェクトコントロールの **InMskHandle**,**OutMskHandle** プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

RectBottom,RectLeft,RectRight,RectTop(イメージキットコントロール／Effect プロパティ)

【機能】
円形にエフェクト処理を施す際の円に外接する四角形、もしくは矩形切り出しの際の四角形の位置を取得または設定します。

【書式】
※**RectBottom** にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Effect**->**RectBottom** [= *int*]
(2)Delphi　*imagekitcontrolname*.**Effect**.**RectBottom** [= *Integer*]

【設定値】
RectBottom は、四角形の下位置(ピクセル)
RectLeft は、四角形の左位置(ピクセル)
RectRight は、四角形の右位置(ピクセル)
RectTop は、四角形の上位置(ピクセル)

【解説】
円に外接する四角形を表す場合は、**SelectMode** プロパティを vikEffectEllipse に設定します。

【値の設定】 実行時

【値の参照】 実行時

SelectMode(イメージキットコントロール／Effect プロパティ)

【機能】
エフェクト処理を施す範囲を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Effect->SelectMode** [= *TVIkAreaSelectMode*]
(2)Delphi　*imagekitcontrolname*.**Effect.SelectMode** [= *TVIkAreaSelectMode*]

【TVIkInAreaSelectMode 型】
ユニット
IkInit
type
TVIkAreaSelectMode = (vikEffectMask, vikEffectAll, vikEffectPolygon, vikEffectEllipse);

【設定値】

値	説明
vikEffectMask	マスクハンドル(**MaskFileName**,**MaskImageHandle** プロパティに設定した値が有効)
vikEffectAll	イメージ全体(**ImageHandle** プロパティに設定した値が有効)
vikEffectPolygon	選択した多角形(メソッドの引数の TPoint 型の配列に設定した値が有効)
vikEffectEllipse	選択した円形(**RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに設定した値が有効)

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikEffectMask, ikEffectAll, ikEffectPolygon, ikEffectEllipse)。

Affine(イメージキットコントロール／Effect メソッド)

【機能】
歪んだラスタイメージを修正します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**Affine**(int Px1, int Py1, int Px2, int Py2, int Px3, int Py3, int Px4, int Py4)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Affine**(Px1, Py1, Px2, Py2, Px3, Py3, Px4, Py4: Integer)

【引数】

名称	内容
Px1～Px4	アフィン変換する四点の X 座標(ピクセル)
Py1～Py4	アフィン変換する四点の Y 座標(ピクセル)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
(Px1,Py1)、(Px2,Py2)、(Px3,Py3)、(Px4,Py4)で囲まれたイメージをその点に内接する四角形に変換します。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

AntiAlias(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージのエッジを滑らかにします。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**AntiAlias**(const TPoint * Points, const int Points_Size)
[*bool =*]*imagekitcontrolname*->**Effect**->**AntiAlias**()
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**AntiAlias**(const Points: array of TPoint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**AntiAlias**

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

AutoSelectImage(イメージキットコントロール／Effect メソッド)

【機能】
指定した RGB 値に近似している部分を選択し、ラスタイメージとマスクを生成します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**AutoSelectImage**(const TPoint * Points, const int Points_Size, short Red, short Green, short Blue, short Mode, short Level))
[*bool =*]*imagekitcontrolname*->**Effect**->**AutoSelectImage**(short Red, short Green, short Blue, short Mode, short Level))
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**AutoSelectImage**(const Points: array of TPoint; Red, Green, Blue, Mode, Level: Smallint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**AutoSelectImage**(Red, Green, Blue, Mode, Level: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。	
Red	選択対象となる赤の値(0～255)
Green	選択対象となる緑の値(0～255)
Blue	選択対象となる青の値(0～255)
Mode	比較モード(0:RGB, 1:CrCb, 2:Y)
Level	許容誤差(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** もしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。
SelectMode プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です(**SelectMode** プロパティが vikEffectMask の場合はエラーになります)。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。

Mode が 0 以外(CrCb:色相 Y:輝度)の場合には、設定された RGB 値を YCrCb に変換して比較します。
Level(許容誤差)は指定した RGB にどの程度近いピクセルを選択するかを制御します。値が 0 の場合は、指定した RGB 値そのものを選択し、値を大きくすると許容誤差が大きくなり、より多くのピクセルを選択します。
成功した場合は、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に、マスクイメージのメモリハンドルは **MaskImageHandle** プロパティに設定されます。指定した RGB 値が選択できなかった場合は黒のイメージが返されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

Blur(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージにぼかしを施します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**Blur**(const TPoint * Points, const int Points_Size, short Level)
[*bool =*]*imagekitcontrolname*->**Effect**->**Blur**(short Level)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Blur**(const Points: array of TPoint; Level: Smallint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Blur**(Level: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位) ※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Level	ぼかしの強さ(1～20 数が大きいほど強い)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Level を大きくすればするほど、イメージはぼやけていきます。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティにに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

Canvas(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージにキャンバス地効果を施します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**Canvas**(const TPoint * Points, const int Points_Size, short Direction, short Depth)
[*bool* =]*imagekitcontrolname*->**Effect**->**Canvas**(short Direction, short Depth)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**Canvas**(const Points: array of TPoint; Direction: Smallint; Depth: Smallint)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**Canvas**(Direction: Smallint; Depth: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位) ※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Direction	目地の方向(0:上から右、1:上から左、2:下から左、3:下から右)
Depth	目地の深さ(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。

Depth が大きいほど彫りが深くなります。
成功した場合は、処理後のラスタイメージのメモリハンドルが **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

CheckSecretImage(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージから透かし情報を取り出します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**CheckSecretImage**(NativeUInt Handle1, NativeUInt Handle2, int Level)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**CheckSecretImage**(Handle1, Handle2: THandle; Level: Integer)

【引数】

名称	内容
Handle1	キーとなるラスタイメージのメモリハンドル(24 ビットイメージが対象)
Handle2	透かしが埋め込まれたラスタイメージのメモリハンドル(24 ビットイメージが対象)
Level	透かしの取り出しレベル(-20～20)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
SetSecretImage メソッドで設定した透かし情報を取得します。
Level には、**SetSecretImage** メソッドで設定した Level 付近、すなわち＋で埋め込んだ場合は埋め込み値より小さめな値を、－で埋め込んだ場合は埋め込み値より大きめな値を設定します。
それ以外の値を設定した場合は、すかし情報が抽出されない場合があります。

成功した場合、処理後のラスタイメージのメモリハンドルは **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定されます(**LayerNo** プロパティによって設定されるプロパティが決まります)。設定されるイメージは、黒の背景に白いすかし情報となります(1 ビットイメージでイメージの大きさは Handle1、Handle2 と同じ)。
引数として与えたメモリハンドルは解放されずにそのまま残ります。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

Chroma(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの彩度を調整します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**Chroma**(const TPoint * Points, const int Points_Size, int Level)
[*bool =*]*imagekitcontrolname*->**Effect**->**Chroma**(int Level)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Chroma**(const Points: array of TPoint; Level: Integer)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Chroma**(Level: Integer)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位) ※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Level	彩度の加減 (-100～1000)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Level が 0 の場合には何も行わず、-100 の場合は見かけ上グレースケールと同じになります。
－に設定すると色は相対的に淡くなり、＋に設定すると鮮やかになります。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
1,4,8 ビットイメージの場合は、**SelectMode** プロパティの値に関わらずイメージ全体に対して処理を行います。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

ConvertColor(イメージキットコントロール／Effect メソッド)

【機能】
メモリ上のラスタイメージの色数を変更します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**ConvertColor**(short PixelType, bool FixedPal, bool Dither, short Level)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**ConvertColor**(PixelType: Smallint; FixedPal, Dither: Boolean; Level: Smallint)

【引数】

名称	内容
PixelType	変更後のピクセル当たりのビット数(1,4,8,16,24,32,40,80) ※4 は 4 ビットカラー、40 は 4 ビットグレー、8 は 8 ビットカラー、80 は 8 ビットグレーとなります。
FixedPal	固定パレットの使用設定(1,4,8 ビットカラーに減色および増色する際に使用) False:イメージに合わせたパレットを作成し使用 True:固定パレットを使用
Dither	ディザリング(誤差拡散法)の設定(1,4,8 ビットカラーおよび 4 ビットグレーに減色する際に使用) False:ディザリングを行わない True:ディザリングを行う
Level	しきい値の設定(1 ビットイメージに減色する際に使用) 0～255

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
1 ビットイメージに減色した場合は、ピクセルの RGB の輝度が Level で設定された値以上の場合はパレット 1 を、それ未満の場合はパレット 0 として処理を行います。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** に設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

CustomFilter(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージに対してユーザ独自のフィルタ処理を施します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**CustomFilter**(const TPoint * Points, const int Points_Size, const int * Matrix, const int Matrix_Size, short ADiv, short Level, bool Red, bool Green, bool Blue)
[*bool =*]*imagekitcontrolname*->**Effect**->**CustomFilter**(const int * Matrix, const int Matrix_Size, short ADiv, short Level, bool Red, bool Green, bool Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**CustomFilter**(const Points: array of TPoint; const Matrix: array of Integer; ADiv, Level: Smallint; Red, Green, Blue: Boolean)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**CustomFilter**(const Matrix: array of Integer; ADiv, Level: Smallint; Red, Green, Blue: Boolean)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位) ※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Matrix	9×9 行列の係数(配列、要素数は 81 で添字は 0～80) ※C++Builder の場合、Matrix_Size に 80 を与えます。
ADiv	行列に対する除数
Level	行列に対する加減(0～255 大きいほど明るい)
Red	赤のプレーンに対する処理(False:しない、True:する)
Green	緑のプレーンに対する処理(False:しない、True:する)
Blue	青のプレーンに対する処理(False:しない、True:する)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
グレースケールのイメージを処理する場合は、Red,Green,Blue で設定された値は無効となります。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

出力するピクセルの計算式は以下のようになります。

$$f = \left\{ \sum_{i=0}^{80} PiCi \right\} \div Div + Level$$

Pi はピクセルの値、Ci は Matrix の係数の値を表します。

簡単なエンボスフィルタの例を示します。

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	−1	0	1	0	0	0
0	0	0	−1	0	1	0	0	0
0	0	0	−1	0	1	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Matrix[0]～Matrix[8]
Matrix[9]～Matrix[17]
Matrix[18]～Matrix[26]
Matrix[27]～Matrix[35]
Matrix[36]～Matrix[44]
Matrix[45]～Matrix[53]
Matrix[54]～Matrix[62]
Matrix[63]～Matrix[71]
Matrix[72]～Matrix[80]

ADiv = 1、Level = 128

上の表の値は Matrix の配列のデータを示します。(左から右へ、上から下へいく毎に添字が増える)

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。
・引数の Matrix が配列型に変更されました。
・C++Builder で使用する場合、Matrix 配列の要素数が引数に追加されました。

CutRectImage(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージから指定した矩形領域を切り出します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Effect**->**CutRectImage**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Effect**.**CutRectImage**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
マスクイメージを必要としない矩形の切り出しに使用できます(マスクイメージが必要な場合は **SelectImage** メソッドをご使用ください)。
処理速度は **SelectImage** メソッドと比較して高速です。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定し(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)、切り取る矩形の位置を **RecLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに設定します。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

Emboss(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージにエンボス(浮き彫り)処理を施します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**Emboss**(const TPoint * Points, const int Points_Size, short MskPattern, short Edge, short Level)
[*bool* =]*imagekitcontrolname*->**Effect**->**Emboss**(short MskPattern, short Edge, short Level)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**Emboss**(const Points: array of TPoint; MskPattern, Edge, Level: Smallint)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**Emboss**(MskPattern, Edge, Level: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

MskPattern	フィルタマスクのパターン(0～7)
Edge	エッジの強調(1～5 大きいほど強い)
Level	背景の輝度(0～255 大きいほど明るい)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
MskPattern によって浮き彫り方向が変わります。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

EndDibAccess(イメージキットコントロール／Effect メソッド)

【機能】
DIB へのアクセス処理を終了します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Effect**->**EndDibAccess**(short DibNo)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Effect**.**EndDibAccess**(DibNo: Smallint)

【引数】

名称	内容
DibNo	**StartDibAccess** メソッドで取得したラスタイメージの管理番号

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
StartDibAccess メソッドと対で使用します。
DibNo は **StartDibAccess** メソッドで取得した値と同じでなければなりません。

処理の流れとしては
StartDibAccess
　|
GetDibPixel, SetDibPixel
　|
EndDibAccess
となります。

コード例については **StartDibAccess** メソッドを参照してください。

※DLL の関数を使用した方が処理速度は速くなります。

GetDibPixel(イメージキットコントロール／Effect メソッド)

【機能】
DIB から指定したピクセルの RGB やパレット番号を取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**GetDibPixel**(short DibNo, int x, int y, Byte &PalNo, Byte &Red, Byte &Green, Byte &Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**GetDibPixel**(DibNo: Smallint; x, y: Integer; var PalNo, Red, Green, Blue: Byte)

【引数】

名称	内容
DibNo	**StartDibAccess** メソッドで取得したラスタイメージの管理番号
x,y	取得するピクセルの X,Y 座標
PalNo	パレット番号を取得する変数
Red	RGB の赤を取得する変数
Green	RGB の緑を取得する変数
Blue	RGB の青を取得する変数

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
DibNo は **StartDibAccess** メソッドで取得した値と同じでなければなりません。
1,4,8 ビットイメージの場合は PalNo に、16,24,32 ビットイメージの場合は Red,Green,Blue に有効な値が設定されます。

処理の流れとしては
StartDibAccess
|
GetDibPixel, SetDibPixel
|
EndDibAccess
となります。

コード例については **StartDibAccess** メソッドを参照してください。

※DLL の関数を使用した方が処理速度は速くなります。

【ImageKit7/8/9/10 ActiveX との違い】
引数の PalNo, Red, Green, Blue が Byte 型に変更されました。

GlassTile(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージにガラスタイル効果を施します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**GlassTile**(const TPoint * Points, const int Points_Size, int xSize, int ySize)
[*bool =*]*imagekitcontrolname*->**Effect**->**GlassTile**(int xSize, int ySize)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**GlassTile**(const Points: array of TPoint; xSize, ySize: Integer)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**GlassTile**(xSize, ySize: Integer)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位) ※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
xSize	ガラスタイルの横サイズ(10～100)
ySize	ガラスタイルの縦サイズ(10～100)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

LayerImage(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの重ね合わせを行います。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**LayerImage**(NativeUInt Handle1, NativeUInt Handle2, short Trans, bool TransColor, short TRed, short TGreen, short TBlue, short BRed, short BGreen, short BBlue, int x, int y, bool Clip)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**LayerImage**(Handle1, Handle2: THandle; Trans: Smallint; TransColor: Boolean; TRed, TGreen, TBlue, BRed, BGreen, BBlue: Smallint; x, y: Integer; Clip: Boolean)

【引数】

名称	内容
Handle1	重ねられるラスタイメージのハンドル(1,4,8,16,24,32 ビットイメージ)
Handle2	重ねるラスタイメージのハンドル(1,4,8,16,24,32 ビットイメージ)
Trans	透かしの度合い (0～255 数が大きいほど Handle2 が生きる)
TransColor	Handle2 に対する透明色の設定(False:なし、True:あり)
TRed	Handle2 に対する透明色 赤(0～255)
TGreen	Handle2 に対する透明色 緑(0～255)
TBlue	Handle2 に対する透明色 青(0～255)
BRed	重ね合わせたイメージの無効領域の背景色 赤(0～255)
Bgreen	重ね合わせたイメージの無効領域の背景色 緑(0～255)
BBlue	重ね合わせたイメージの無効領域の背景色 青(0～255)
x,y	貼り付け座標(ピクセル)
Clip	クリッピングの有無(False:なし、True:あり)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Handle2 のイメージの左上の点を、Handle1 のイメージの x,y の位置に合わせて貼り付けを行います。
Clip が True の時は、Handle1 のイメージのサイズより大きくなった場合に、はみでた領域をカットします。
TransColor が True の場合は、TRed・TGreen・TBlue で設定された色を透明色として使用します。
アルファチャンネルを考慮して重ね合わせをする場合は、**LayerAlphaChannel** プロパティを True にして、かつ Handle2 に有効なアルファチャンネルが入った RGBA の 32 ビットイメージを与える必要があります。
なお、**LayerAlphaChannel** プロパティを True に設定した場合、Trans は無効になります。
成功した場合、処理後のラスタイメージのメモリハンドルは **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定されます(**LayerNo** プロパティによって設定されるプロパティが決まります)。
引数として与えたメモリハンドルは解放されずにそのまま残ります。

出力されるイメージのビット数の説明
- Handle1 と Handle2 のビット数が共に 8 ビット以下で、かつ同じビット数であり、かつ同じパレット数と並びの場合は入力イメージのビット数と同じになります。ただし、Handle1 と Handle2 が 8 ビットグレースケールではなく Trans が 0 か 255 以外の場合は 24 ビットイメージになります。
- Handle1 と Handle2 のビット数が共に 8 ビット以下でも、ビット数が違うか、ビット数が同じでもパレット数と並びが違う場合は 24 ビットイメージになります。
- Handle1 と Handle2 のどちらかが 16 ビット以上の場合は、大きいビット数になります。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

LayerImage メソッドは以前の ImageKit で提供していたエフェクトコントロールの **Layer** メソッドに相当します。

Lens（イメージキットコントロール／Effect メソッド）

【機能】
ラスタイメージにレンズ効果を施します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**Lens**(const TPoint * Points, const int Points_Size, double Refract, bool BackColor, short Red, short Green, short Blue)
[*bool =*]*imagekitcontrolname*->**Effect**->**Lens**(double Refract, bool BackColor, short Red, short Green, short Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Lens**(const Points: array of TPoint; Refract: Double; BackColor: Boolean; Red, Green, Blue: Smallint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Lens**(Refract: Double; BackColor: Boolean; Red, Green, Blue: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列（座標はピクセル単位）

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

Refract	レンズの倍率（1.0 以上）
BackColor	レンズの外側の背景色の描画設定 ［ True：塗る　False：塗らない］
Red	レンズの外側の背景色　赤(0～255)
Green	レンズの外側の背景色　緑(0～255)
Blue	レンズの外側の背景色　青(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します（8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります）。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列（もしくは NULL）を与えるか、もしくは Points が不要なメソッドをご利用ください。
BackColor が True の時、Red,Green,Blue が有効になります。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

MakeRGBAImage（イメージキットコントロール／Effect メソッド）

【機能】
RGB と A から RGBA の 32 ビットイメージを作成します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Effect**->**MakeRGBAImage**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Effect**.**MakeRGBAImage**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
RGB イメージのメモリハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ（1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります）に設定し、A（アルファチャンネル）イメージのメモリハンドルを **MaskImageHandle** プロパティ（1,8 ビットグレースケールイメージ）に設定します。
成功した場合、RGBA の 32 ビットイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に％形式で表示されます。

Mosaic(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージにモザイク処理を施します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**Mosaic**(const TPoint * Points, const int Points_Size, int AWidth, int AHeight)
[*bool =*]*imagekitcontrolname*->**Effect**->**Mosaic**(int AWidth, int AHeight)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Mosaic**(const Points: array of TPoint; AWidth, AHeight: Integer)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Mosaic**(AWidth, AHeight: Integer)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位) ※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
AWidth	モザイクの幅(ピクセル)
AHeight	モザイクの高さ(ピクセル)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
AWidth と AHeight が共に 1 以下の場合はモザイク処理を行いません。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

MotionBlur(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージにモーションぼかし効果を施します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**MotionBlur**(const TPoint * Points, const int Points_Size, short Mode, short ALength, int Angle)
[*bool =*]*imagekitcontrolname*->**Effect**->**MotionBlur**(short Mode, short ALength, int Angle)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**MotionBlur**(const Points: array of TPoint; Mode, ALength: Smallint; Angle: Integer)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**MotionBlur**(Mode, ALength: Smallint; Angle: Integer)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。	
Mode	ぼかしの種類(0:線形, 1:放射状, 2:拡大)
ALength	ぼかしの度合い(長さ、0～255)
Angle	ぼかしの度合い(角度、1 度単位)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

OilPaint(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージに油絵風効果を施します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**OilPaint**(const TPoint * Points, const int Points_Size, short Mode, short MaskSize)
[*bool* =]*imagekitcontrolname*->**Effect**->**OilPaint**(short Mode, short MaskSize)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**OilPaint**(const Points: array of TPoint; Mode, MaskSize: Smallint)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**OilPaint**(Mode, MaskSize: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
	※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Mode	油絵の質感モード(0:RGB 別処理, 1:輝度別処理)
MaskSize	筆のサイズ (1～100)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

Outline(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージから輪郭部分を抽出します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**Outline**(const TPoint * Points, const int Points_Size, short Plane, short Operator, short Level)
[*bool* =]*imagekitcontrolname*->**Effect**->**Outline**(short Plane, short Operator, short Level)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**Outline**(const Points: array of TPoint; Plane, Operator, Level: Smallint)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**Outline**(Plane, Operator, Level: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
	※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Plane	処理するプレーン(0:Y,1:R,2:G,3:B)
Operator	行列変数のパターン(0:差分, 1:Roberts, 2:Sobel)
Level	輪郭線の明るさ(数が大きいほど明るい 0～100)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
通常は Plane を 0 として輝度の違いを輪郭として認識しますが、1 以上を設定することにより各プレーンの値の差を輪郭として認識することもできます。輪郭抽出の効果は Operator により異なります。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

Panorama(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージのパノラマ合成を行います。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**Panorama**(NativeUInt Handle1, NativeUInt Handle2, short CutMode, int Px11, int Py11, int Px12, int Py12, int Px21, int Py21, int Px22, int Py22, short Red, short Green, short Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Panorama**(Handle1, Handle2: THandle; CutMode:Smallint; Px11, Py11, Px12, Py12, Px21, Py21, Px22, Py22: Integer; Red, Green, Blue: Smallint)

【引数】

名称	内容
Handle1	基となるラスタイメージのハンドル(24 ビットイメージ)
Handle2	貼り付けるラスタイメージのハンドル(24 ビットイメージ)
CutMode	Handle2 の基準点に対してカットする方向を設定(0:なし 1:左 2:右)
Px11,Py11	Handle1 のイメージ上の基準点 1(ピクセル)
Px12,Py12	Handle1 のイメージ上の基準点 2(ピクセル)
Px21,Py21	Handle2 のイメージ上の基準点 1(ピクセル)
Px22,Py22	Handle2 のイメージ上の基準点 2(ピクセル)
Red	重ね合わせたイメージの無効領域の背景色 赤(0～255)
Green	重ね合わせたイメージの無効領域の背景色 緑(0～255)
Blue	重ね合わせたイメージの無効領域の背景色 青(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Handle1 の Px11、Py11、Px12、Py12 で指定した座標に、Handle2 の Px21、Py21、Px22、Py22 で指定した座標を合わせて、CutMode の値により Handle2 のイメージをそのまま合成するかどうかを設定し、Handle1 に Handle2 のイメージを合成します。(Handle2 の基準点が Handle1 の基準点に合わさるような形になります。)
CutMode は基準点 1 を上に基準点 2 を下にした場合の、カットする方向を示しています。

成功した場合、処理後のラスタイメージのメモリハンドルは **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定されます(**LayerNo** プロパティによって設定されるプロパティが決まります)。
設定されるイメージは、24 ビットイメージとなります。引数として与えたメモリハンドルは解放されずにそのまま残ります。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

PasteImage(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの貼り付けを行います。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**PasteImage**(NativeUInt AHandle, int Angle, bool TurnX, bool TurnY, short Trans, bool TransColor, short TransRed, short TransGreen, short TransBlue, short BackRed, short BackGreen, short BackBlue, int x, int y, bool Clip)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**PasteImage**(AHandle: THandle; Angle: Integer; TurnX, TurnY: Boolean; Trans: Smallint; TransColor: Boolean; TransRed, TransGreen, TransBlue, BackRed, BackGreen, BackBlue: Smallint; x, y: Integer; Clip: Boolean)

【引数】

名称	内容
AHandle	貼り付けられるラスタイメージのハンドル(1,4,8,16,24,32 ビットイメージ)
Angle	回転角度(-35999～35999　1/100 度単位)
TurnX	X 方向反転の有無 (False:反転なし True:反転あり)
TurnY	Y 方向反転の有無 (False:反転なし True:反転あり)
Trans	透かしの度合い (0～255 数が大きいほど **LayerNo** プロパティの示すイメージが生きる)
TransColor	**LayerNo** プロパティの示すイメージに対する透明色の設定(False:なし、True:あり)
TransRed	**LayerNo** プロパティの示すイメージに対する透明色 赤(0～255)
TransGreen	**LayerNo** プロパティの示すイメージに対する透明色 緑(0～255)
TransBlue	**LayerNo** プロパティの示すイメージに対する透明色 青(0～255)
BackRed	貼り付けたイメージの無効領域の背景色 赤(0～255)
BackGreen	貼り付けたイメージの無効領域の背景色 緑(0～255)
BackBlue	貼り付けたイメージの無効領域の背景色 青(0～255)
x,y	貼り付け座標(ピクセル)
Clip	クリッピングの有無(False:なし、True:あり)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
Angle、TurnX、TurnY は **LayerNo** プロパティの示すプロパティと **MaskImageHandle** プロパティに対して有効です。
x,y は AHandle の左上座標(0,0)として、**LayerNo** プロパティの示すプロパティと **MaskImageHandle** プロパティの指すイメージの中心点を AHandle のどの座標に貼り付けるかを指定します。
Clip が True の時は、AHandle のイメージのサイズより大きくなった場合に、はみでた領域をカットします。
TransColor が True の場合は、TransRed,TransGreen,TransBlue で設定された色を透明色として使用します。

成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。
引数として与えたメモリハンドルは解放されずにそのまま残ります。

出力されるイメージのビット数の説明
- ImageHandle と AHandle のビット数が共に 8 ビット以下で、かつ同じビット数であり、かつ同じパレット数と並びの場合は入力イメージのビット数と同じになります。ただし、ImageHandle と AHandle が 8 ビットグレースケールではなく Trans が 0 か 255 以外の場合は 24 ビットイメージになります。
- ImageHandle と AHandle のビット数が共に 8 ビット以下でも、ビット数が違うか、ビット数が同じでもパレット数と並びが違う場合は 24 ビットイメージになります。
- ImageHandle と AHandle のどちらかが 16 ビット以上の場合は、大きいビット数になります。

Caption,Message,ButtonName プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

RedEyeRemoval(イメージキットコントロール／Effect メソッド)

【機能】
赤目を指定した色に補正します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**RedEyeRemoval**(const TPoint * Points, const int Points_Size, short Red, short Green, short Blue, short Error)
[*bool =*]*imagekitcontrolname*->**Effect**->**RedEyeRemoval**(short Red, short Green, short Blue, short Error)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**RedEyeRemoval**(const Points: array of TPoint; Red, Green, Blue, Error: Smallint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**RedEyeRemoval**(Red, Green, Blue, Error: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

Red	補正後の赤(0～255)
Green	補正後の緑(0～255)
Blue	補正後の青(0～255)
Error	補正する赤目の誤差範囲(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Error は値が大きい程より赤に近い色を補正し、小さい程補正する色の範囲が広がります。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

RemoveNoise(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージのノイズを除去します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**RemoveNoise**(const TPoint * Points, const int Points_Size, short Mode, short Level)
[*bool* =]*imagekitcontrolname*->**Effect**->**RemoveNoise**(short Mode, short Level)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**RemoveNoise**(const Points: array of TPoint; Mode, Level: Smallint)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**RemoveNoise**(Mode, Level: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位) ※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Mode	処理モード(0:輝度値のメディアン法,1:RGB 別のノイズ除去) 定数(vikRemoveNoiseGray = 0, vikRemoveNoiseRGB = 1)を使用することも可能です。
Level	RGB でのノイズ除去レベル(0～255、Mode が 1 の時に有効)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Mode が 1 の場合に Level で設定された値が有効になります。Level を高く設定するほどノイズを除去することができます。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

【ImageKit7/8 ActiveX/VCL との違い】
白黒 2 値画像に対応しました(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに白黒 2 値画像を設定可能)。

Resize(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージのサイズを変更します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Effect**->**Resize**(int AWidth, int AHeight, bool Mode)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Effect.Resize**(AWidth, AHeight: Integer; Mode: Boolean)

【引数】

名称	内容
AWidth	処理後のイメージの幅(ピクセル)
AHeight	処理後のイメージの高さ(ピクセル)
Mode	補間の有無(False:なし、True:あり) True の場合、拡大時は線形補間を行い、縮小時は平均を取り出力 8 ビットグレースケール,16,24,32 ビットイメージが対象となり、それ以外は無視される

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
基のイメージを、AWidth,AHeight で指定した新しいサイズになるように自動的に縮小、拡大してサイズを変更します。
Mode が False の場合、基のイメージより大きくするとその拡大率に比例してイメージが粗くなります。

処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。**MaskImageHandle** プロパティに有効な値を設定した場合は、その結果として処理後のマスクハンドルが **MaskImageHandle** プロパティに設定されます。
Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

RGBGamma(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの RGB 値のガンマ補正を行います。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**RGBGamma**(const TPoint * Points, const int Points_Size, double Red, double Green, double Blue)
[*bool =*]*imagekitcontrolname*->**Effect**->**RGBGamma**(double Red, double Green, double Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**RGBGamma**(const Points: array of TPoint; Red, Green, Blue: Double)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**RGBGamma**(Red, Green, Blue: Double)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

Red	Red のガンマ係数
Green	Green のガンマ係数
Blue	Blue のガンマ係数

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理後のピクセル値を y、基のピクセル値を x とすると
(1)Bright > 0 の場合

$$y = (x/255)^{Bright+1} \times 255$$

(2)Bright < 0 の場合

$$y = (x/255)^{-1/Bright-1} \times 255$$

となります。

Bright は Red,Green,Blue のいずれかになります。Red,Green,Blue が 0 の場合は処理の前後で変化ありません。
Red,Green,Blue が＋の場合はレベルが下がり、－の場合はレベルが上がります。

処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
1,4,8 ビットイメージの場合は、**SelectMode** プロパティの値に関わらずイメージ全体に対して処理を行います。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアロ

グボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

RGBLevel(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの RGB 値の加減処理を行います。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**RGBLevel**(const TPoint * Points, const int Points_Size, int Red, int Green, int Blue)
[*bool =*]*imagekitcontrolname*->**Effect**->**RGBLevel**(int Red, int Green, int Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**RGBLevel**(const Points: array of TPoint; Red, Green, Blue: Integer)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**RGBLevel**(Red, Green, Blue: Integer)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
	※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Red	Red の加減(-255～255)
Green	Green の加減(-255～255)
Blue	Blue の加減(-255～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Red、Green、Blue のそれぞれの値が大きくなると該当する成分が明るくなり、小さくなると該当する成分が暗くなります。
また、Red、Green、Blue が 0 の場合は処理の前後で変化ありません。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
1,4,8 ビットイメージの場合は、**SelectMode** プロパティの値に関わらずイメージ全体に対して処理を行います。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

RGBRev(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの RGB 値の反転処理を行います。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**RGBRev**(const TPoint * Points, const int Points_Size, bool Red, bool Green, bool Blue)
[*bool* =]*imagekitcontrolname*->**Effect**->**RGBRev**(bool Red, bool Green, bool Blue)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**RGBRev**(const Points: array of TPoint; Red, Green, Blue: Boolean)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**RGBRev**(Red, Green, Blue: Boolean)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
	※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Red	Red の反転(False: 反転なし, True: 反転あり)
Green	Green の反転(False: 反転なし, True: 反転あり)
Blue	Blue の反転(False: 反転なし, True: 反転あり)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Red、Green、Blue を True にするとそれぞれの RGB 値を反転します。
例えば、Red が 255 の場合は 0 となります。(Red = 255 - Red)

処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
1,4,8 ビットイメージの場合は、**SelectMode** プロパティの値に関わらずイメージ全体に対して処理を行います。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

RGBSpline（イメージキットコントロール／Effect メソッド）

【機能】
ラスタイメージの RGB 値のスプライン補正を行います。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**RGBSpline**(const TPoint * Points, const int Points_Size, const TPoint * spPoints, const int spPoints_Size, bool Red, bool Green, bool Blue)
[*bool* =]*imagekitcontrolname*->**Effect**->**RGBSpline**(onst TPoint * spPoints, const int spPoints_Size, bool Red, bool Green, bool Blue)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**RGBSpline**(const Points: array of TPoint; const spPoints: array of TPoint; Red, Green, Blue: Boolean)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**RGBSpline**(const spPoints: array of TPoint; Red, Green, Blue: Boolean)

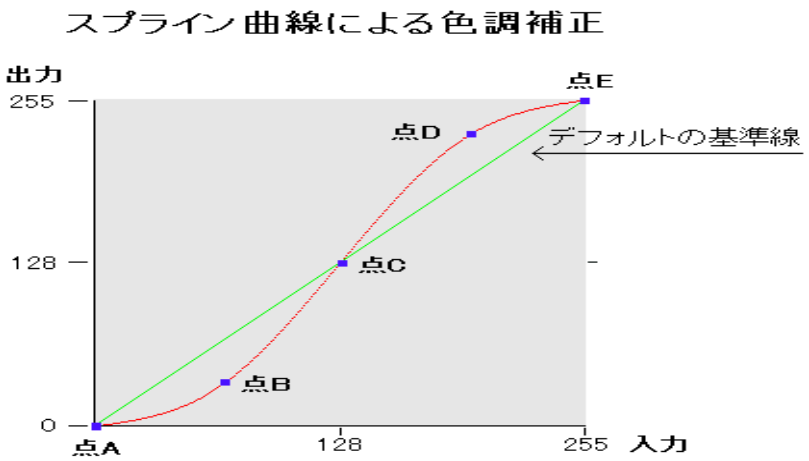
【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列（座標はピクセル単位）
※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。	
spPoints	スプラインの通過 x,y 座標を指定する配列（座標は 0～255 の間） （値の設定方法は Points と同じ、配列の要素数は 3～10 が有効）
※C++Builder の場合、spPoints の要素数-1 を spPoints_Size に与えます。	
Red	赤の成分に対して処理を行うかどうかの設定（False:しない True:する）
Green	緑の成分に対して処理を行うかどうかの設定（False:しない True:する）
Blue	青の成分に対して処理を行うかどうかの設定（False:しない True:する）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
入力されたイメージデータの RGB 値に対してスプライン曲線により変化させることができます。

「使用例」
spx,spy の値　点A(0,0)　点B(64,30)　点C(128,128)　点D(192,225)　点E(255,255)
spPoints の値　5
・作成される曲線は spx,spy を通過するスプライン曲線を描きます。（最大 10 ポイントまで）



処理対象となるイメージハンドルを ImageHandle プロパティもしくは Layer[LayerNo].ImageHandle プロパティに設定します（1,4,8,16,24,32 ビットイメージが対象で LayerNo プロパティによって処理されるイメージハンドルが決まります）。必要に応じてマスクイメージのハンドルを MaskImageHandle プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、SelectMode プロパティに vikEffectMask を MaskImageHandle プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、SelectMode プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、SelectMode プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、SelectMode プロパティに vikEffectEllipse を設定し RectLeft,RectTop,RectRight,RectBottom プロパティに有効な値を設定します。SelectMode プロパティが vikEffectPolygon と vikEffectEllipse の場合には InOut プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列（もしくは NULL）を与えるか、もしくは Points が不要なメソッドをご利用ください。
1,4,8 ビットイメージの場合は、SelectMode プロパティの値に関わらずイメージ全体に対して処理を行います。
成功した場合、処理後のラスタイメージのメモリハンドルは LayerNo プロパティの示すプロパティ（ImageHandle プロパティもしくは Layer[LayerNo].ImageHandle プロパティ）に設定されます。

Caption,Message,ButtonName プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

Ripple（イメージキットコントロール／Effect メソッド）

【機能】
ラスタイメージに波紋効果を施します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**Ripple**(const TPoint * Points, const int Points_Size, double Amp, double Wavelen, double Phase, bool BackColor, bool Reflect, short Red, short Green, short Blue)
[*bool =*]*imagekitcontrolname*->**Effect**->**Ripple**(double Amp, double Wavelen, double Phase, bool BackColor, bool Reflect, short Red, short Green, short Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Ripple**(const Points: array of TPoint; Amp, Wavelen, Phase: Double; BackColor, Reflect: Boolean; Red, Green, Blue: Smallint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Ripple**(Amp, Wavelen, Phase: Double; BackColor, Reflect: Boolean; Red, Green, Blue: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列（座標はピクセル単位） ※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Amp	波の振幅の大きさ
Wavelen	波長の長さ
Phase	波の位相
BackColor	空白部分の設定 [False:背景色で塗る,True:隣接するピクセルで塗る]
Reflect	波の反射設定 [False:しない,True:する]
Red	背景色 赤(0～255)
Green	背景色 緑(0～255)
Blue	背景色 青(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します（8ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります）。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列（もしくは NULL）を与えるか、もしくは Points が不要なメソッドをご利用ください。
Red,Green,Blue は BackColor が False の場合に有効です。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

Rotation(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージを回転させます。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**Rotation**(int Angle, bool TurnX, bool TurnY, bool Mode, short Red, short Green, short Blue, bool Clip)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Rotation**(Angle: Integer; TurnX, TurnY, Mode: Boolean; Red, Green, Blue: Smallint; Clip: Boolean)

【引数】

名称	内容
Angle	回転角度（-35999～35999　1/100 度単位）
TurnX	X 方向反転の有無（False:反転なし True:反転あり）
TurnY	Y 方向反転の有無（False:反転なし True:反転あり）
Mode	補間の有無(False:なし、True:あり) True の場合は線形補間を行う 8 ビットグレースケール,16,24,32 ビットイメージが補間の対象となる
Red	イメージの背景色の赤（0～255）
Green	イメージの背景色の緑（0～255）
Blue	イメージの背景色の青（0～255）
Clip	クリッピングの有無(False:なし、True:あり)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Angle の値が＋の場合は反時計周りに回転し、－の場合は時計回りに回転します。
Clip が True の時は、基のイメージのサイズより大きくなった場合に、はみでた領域をカットします。

処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。**MaskImageHandle** プロパティに有効な値を設定した場合は、その結果として処理後のマスクハンドルが **MaskImageHandle** プロパティに設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

SelectImage(イメージキットコントロール／Effect メソッド)

【機能】
選択範囲を基にラスタイメージとマスクイメージの部分生成を行います。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**SelectImage**(const TPoint * Points, const int Points_Size, short Red, short Green, short Blue)
[*bool =*]*imagekitcontrolname*->**Effect**->**SelectImage**(short Red, short Green, short Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**SelectImage**(const Points: array of TPoint; Red, Green, Blue: Smallint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**SelectImage**(Red, Green, Blue: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
	※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Red	選択外領域の赤の値(0～255)
Green	選択外領域の緑の値(0～255)
Blue	選択外領域の青の値(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。
SelectMode プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
選択した領域を含む最小限な矩形を生成して、**LayerNo** プロパティが示すプロパティと **MaskImageHandle** プロパティにそれぞれのイメージハンドルをセットします。
LayerNo プロパティが示すプロパティにセットされるイメージの選択した領域外は、Red,Green,Blue で指定された RGB 値により塗り潰されます。**MaskImageHandle** プロパティにセットされるマスクイメージは、選択した領域が白で塗り潰されます。(領域外は黒)
Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

矩形の切り出しでマスクイメージが不要な場合は、**CutRectImage** メソッドを使用することにより処理速度が向上します。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

SetDibPixel(イメージキットコントロール／Effect メソッド)

【機能】
指定したピクセルの RGB やパレット番号を DIB に設定します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**SetDibPixel**(short DibNo, int x, int y, Byte PalNo, Byte Red, Byte Green, Byte Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**SetDibPixel**(DibNo: Smallint; x, y: Integer; PalNo, Red, Green, Blue: Byte)

【引数】

名称	内容
DibNo	**StartDibAccess** メソッドで取得したラスタイメージの管理番号
x,y	設定するピクセルの X,Y 座標
PalNo	設定するパレット番号
Red	設定する RGB の赤
Green	設定する RGB の緑
Blue	設定する RGB の青

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
DibNo は **StartDibAccess** メソッドで取得した値と同じでなければなりません。
1,4,8 ビットイメージの場合は PalNo に、16,24,32 ビットイメージの場合は Red,Green,Blue に有効な値を設定します。

処理の流れとしては
StartDibAccess
|
GetDibPixel, SetDibPixel
|
EndDibAccess
となります。

コード例については **StartDibAccess** メソッドを参照してください。

※DLL の関数を使用した方が処理速度は速くなります。

【ImageKit7/8/9/10 ActiveX との違い】
引数の PalNo, Red, Green, Blue が Byte 型に変更されました。

SetGray(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージをグレースケールに変換します。

【書式】
(1)C++Builder　[*bool* =]*imagekitcontrolname*->**Effect**->**SetGray**()
(2)Delphi　[*Boolean* =]*imagekitcontrolname*.**Effect**.**SetGray**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。
なお、**SetGray** メソッドはビット数をそのままに処理を行いますので、8 ビットグレースケールに変換する場合は **ConvertColor** メソッドをご使用ください。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

SetSecretImage(イメージキットコントロール／Effect メソッド)

【機能】
透かし情報をラスタイメージに埋め込みます。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**SetSecretImage**(NativeUInt Handle1, NativeUInt Handle2, const UnicodeString Text, const UnicodeString FName, short FSize, bool Bold, bool Italic, bool Underline, bool StrikeOut, short Direction, int Angle, int Level, int ALeft, int ATop, int ARight, int ABottom)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**SetSecretImage**(Handle1, Handle2: THandle; const Text: string; const FName: string; FSize: Smallint; Bold, Italic, Underline, StrikeOut: Boolean; Direction: Smallint; Angle, Level, ALeft, ATop, ARight, ABottom: Integer)

【引数】

名称	内容
Handle1	基となるラスタイメージのハンドル(24 ビットイメージ)
Handle2	埋め込むラスタイメージのハンドル(1 ビットイメージ)
Text	埋め込む文字列(ヌルを含めて 128 文字以下)
FName	フォントの名称(ヌルを含めて 32 文字以下)
FSize	フォントのサイズ(ポイントで指定)
Bold	False 以外のとき、ボールド体のフォント
Italic	False 以外のとき、イタリック体のフォント
Underline	False 以外のとき、下線付きのフォント
StrikeOut	False 以外のとき、打ち消し線付きのフォント
Direction	文字列の方向(0:左から右 1:右から左 2:上から下 3:下から上)
Angle	文字の回転角度(0, 90, 180, 270)
Level	埋め込む輝度のレベル(-20～20)
ALeft	Handle1 のイメージに対して文字列または Handle2 のイメージを埋め込む左上の X 座標
ATop	Handle1 のイメージに対して文字列または Handle2 のイメージを埋め込む左上の Y 座標
ARight	Handle1 のイメージに対して文字列を埋め込む右下の X 座標
ABottom	Handle1 のイメージに対して文字列を埋め込む右下の Y 座標

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Level を大きくすると透かし情報を埋め込んだところが明るくなり、小さくすると透かし情報を埋め込んだところが暗くなります。
Underline と StrikeOut は、False 以外でも Direction や Angle の値により無効となる場合があります。
ALeft,ATop,ARight,ABottom はピクセル単位で設定します。

Handle2 が 0 でない場合は、Handle2 のイメージが埋め込む透かし情報となります。
Handle2 が 0 の場合は、Text で指定した文字列が透かし情報となり、ALeft,ATop,ARight,ABottom で指定した範囲に収まるようにセットします。
128 バイト以上の文字情報をセットする、もしくは複数の場所に透かし情報を埋め込みたい場合は、Handle2 のイメージをそのように作成して対応してください。

成功した場合、処理後のラスタイメージのメモリハンドルは **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定されます(LayerNo プロパティによって設定されるプロパティが決まります)。
設定されるイメージは、24 ビットイメージとなります。
引数として与えたメモリハンドルは解放されずにそのまま残ります。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

Sharp(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの輪郭を強調してシャープにします。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**Sharp**(const TPoint * Points, const int Points_Size, short Level)
[*bool =*]*imagekitcontrolname*->**Effect**->**Sharp**(short Level)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Sharp**(const Points: array of TPoint; Level: Smallint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**Sharp**(Level: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

Level	シャープネスの強さ(0～100 数が大きいほど強い)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Level を大きくすればするほど、イメージはくっきりしていきます。
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

SplitRGBAImage（イメージキットコントロール／Effect メソッド）

【機能】
RGBA の 32 ビットイメージから RGB と A のイメージに分割します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Effect**->**SplitRGBAImage**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Effect**.**SplitRGBAImage**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
RGBA の 32 ビットイメージのメモリハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します（**LayerNo** プロパティによって処理されるイメージハンドルが決まります）。
成功した場合、RGB の 24 ビットイメージのメモリハンドルが **LayerNo** プロパティの示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に、A（アルファチャンネル）の 8 ビットグレースケールイメージのメモリハンドルが **MaskImageHandle** プロパティに設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

StartDibAccess（イメージキットコントロール／Effect メソッド）

【機能】
DIB へのアクセス処理を開始します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Effect**->**StartDibAccess**(NativeUInt hDib, short &DibNo)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Effect**.**StartDibAccess**(hDib: THandle; var DibNo: Smallint)

【引数】

名称	内容
hDib	ラスタイメージのメモリハンドル（1,4,8,16,24,32 ビットイメージ）
DibNo	hDIB の管理番号を取得する変数

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
DibNo は **GetDibPixel**,**SetDibPixel**,**EndDibAccess** メソッドで使用する hDib の管理番号となります。

処理の流れとしては
StartDibAccess
　　|
GetDibPixel, SetDibPixel
　　|
EndDibAccess
となります。

※DLL の関数を使用した方が処理速度は速くなります。

表示イメージ（1,16,24,32 ビット）をビット反転するコード例：
(1)C++Buildr

```
bool Ret;
short DibNo;
int i, j;
Byte PalNo, Red, Green, Blue;

VImageKit1->LayerNo = -1;
Ret = VImageKit1->GetImageType();
if (Ret == false) return;
Ret = VImageKit1->Effect->StartDibAccess(VImageKit1->ImageHandle, DibNo);
if (Ret == false) return;

for (i = 0; i < VImageKit1->ImageHeight; i++)
{
   for (j = 0; j < VImageKit1->ImageWidth; j++)
   {
      Ret = VImageKit1->Effect->GetDibPixel(DibNo, j, i, PalNo, Red, Green, Blue);
      if (VImageKit1->BitCount == 1)
      {
         if (PalNo != 0)
            PalNo = 0;
         else
            PalNo = 1;
      } else if (VImageKit1->BitCount > 8) {
         Blue = 255 - Blue;
```

```
            Green = 255 - Green;
            Red = 255 - Red;
          }
          Ret = VImageKit1->Effect->SetDibPixel(DibNo, j, i, PalNo, Red, Green, Blue);
        }
      }
      Ret = VImageKit1->Effect->EndDibAccess(DibNo);

      VImageKit1->Refresh();
```

(2)Delphi

```
    Ret: Boolean;
    DibNo: Smallint;
    i, j: Longint;
    PalNo, Red, Green, Blue: Byte;

    VImageKit1.LayerNo := -1;
    Ret := VImageKit1.GetImageType();
    if Ret = False then Exit;
    Ret := VImageKit1.Effect.StartDibAccess(ImageKit1.ImageHandle, DibNo);
    if Ret = False then Exit;

    for i := 0 to VImageKit1.ImageHeight - 1 do
    begin
      for j := 0 to VImageKit1.ImageWidth - 1 do
      begin
        Ret := VImageKit1.Effect.GetDibPixel(DibNo, j, i, PalNo, Red, Green, Blue);
        if VImageKit1.BitCount = 1 then
        begin
          if PalNo <> 0 then
            PalNo := 0
          else
            PalNo := 1;
        end
        else if VImageKit1.BitCount > 8 then
        begin
          Blue := 255 - Blue;
          Green := 255 - Green;
          Red := 255 - Red;
        end;
        Ret := VImageKit1.Effect.SetDibPixel(DibNo, j, i, PalNo, Red, Green, Blue);
      end;
    end;
    Ret := VImageKit1.Effect.EndDibAccess(DibNo);

    VImageKit1.Refresh;
```

UnifyColor(イメージキットコントロール／Effect メソッド)

【機能】
色のばらつきを修正します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**UnifyColor**(const TPoint * Points, const int Points_Size, short Red, short Green, short Blue, short Error)
[*bool =*]*imagekitcontrolname*->**Effect**->**UnifyColor**(short Red, short Green, short Blue, short Error)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**UnifyColor**(const Points: array of TPoint; Red, Green, Blue, Error: Smallint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**UnifyColor**(Red, Green, Blue, Error: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

Red	修正対象となる赤(0～255)
Green	修正対象となる緑(0～255)
Blue	修正対象となる青(0～255)
Error	各 RGB の誤差範囲(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
指定した Red,Green,Blue に対して±Error の範囲の色を Red,Green,Blue に置き換えます。
例えば Red,Green,Blue に 128、Error に 1 を設定した場合は、Red,Green,Blue の 127 から 129 の範囲の色をそれぞれを 128 に置き換えます。

処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

Waves(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージにさざ波効果を施します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**Waves**(const TPoint * Points, const int Points_Size, short Direction, short WaveType, int Period, int Amp, double Phase, short Edges, short Red, short Green, short Blue)
[*bool* =]*imagekitcontrolname*->**Effect**->**Waves**(short Direction, short WaveType, int Period, int Amp, double Phase, short Edges, short Red, short Green, short Blue)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**Waves**(const Points: array of TPoint; Direction, WaveType: Smallint; Period, Amp: Integer; Phase: Double; Edges: Smallint; Red, Green, Blue: Smallint)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**Waves**(Direction, WaveType: Smallint; Period, Amp: Integer; Phase: Double; Edges: Smallint; Red, Green, Blue: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位) ※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Direction	波の方向(0:横,1:縦)
WaveType	波の種類(0:sin,1:saw)
Period	波の波長(0 以上)
Amp	波の大きさ(0 以上)
Phase	波の位相(0.0～1.0)
Edges	エッジの種類(0:背景色で塗る, 1:隣接したピクセルで埋める, 2:丸める)
Red	背景色の赤(0～255)
Green	背景色の緑(0～255)
Blue	背景色の青(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** に設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
Red,Green,Blue は Edges が 0 の場合に有効です。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

WhirlPinch(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージにねじりつまみ効果を施します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**WhirlPinch**(const TPoint * Points, const int Points_Size, double Whirl, double Pinch, double Radius, short Red, short Green, short Blue)
[*bool =*]*imagekitcontrolname*->**Effect**->**WhirlPinch**(double Whirl, double Pinch, double Radius, short Red, short Green, short Blue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**WhirlPinch**(const Points: array of TPoint; Whirl, Pinch, Radius: Double; Red, Green, Blue: Smallint)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**WhirlPinch**(Whirl, Pinch, Radius: Double; Red, Green, Blue: Smallint)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

Whirl	ねじる角度(度単位)
Pinch	つまみの度合い
Radius	ねじりの大きさ(0 以上)
Red	背景色の赤(0～255)
Green	背景色の緑(0～255)
Blue	背景色の青(0～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(8 ビットグレースケール,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。

マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

YCCGamma(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの YCrCb 値のガンマ補正を行います。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**YCCGamma**(const TPoint * Points, const int Points_Size, double Yb, double Cr, double Cb)
[*bool =*]*imagekitcontrolname*->**Effect**->**YCCGamma**(double Yb, double Cr, double Cb)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**YCCGamma**(const Points: array of TPoint; Yb, Cr, Cb: Double)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**YCCGamma**(Yb, Cr, Cb: Double)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

Yb	輝度値のガンマ係数
Cr	色相 Cr のガンマ係数
Cb	色相 Cb のガンマ係数

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理後のピクセル値を y、基のピクセル値を x とすると
(1)Bright > 0 の場合

$$y = (x/255)^{Bright+1} \times 255$$

(2)Bright < 0 の場合

$$y = (x/255)^{-1/Bright-1} \times 255$$

となります。

Bright は Yb,Cr,Cb のいずれかになります。Yb,Cr,Cb が 0 の場合は処理の前後で変化ありません。
Yb,Cr,Cb が＋の場合はレベルが下がり、－の場合はレベルが上がります。

処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
1,4,8 ビットイメージの場合は、**SelectMode** プロパティの値に関わらずイメージ全体に対して処理を行います。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

YCCLevel(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの YCrCb 値の加減処理を行います。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**YCCLevel**(const TPoint * Points, const int Points_Size, int Yb, int Cr, int Cb)
[*bool* =]*imagekitcontrolname*->**Effect**->**YCCLevel**(int Yb, int Cr, int Cb)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**YCCLevel**(const Points: array of TPoint; Yb, Cr, Cb: Integer)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**YCCLevel**(Yb, Cr, Cb: Integer)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
	※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。
Yb	輝度値の加減(-255～255)
Cr	色相 Cr の加減(-255～255)
Cb	色相 Cb の加減(-255～255)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Yb,Cr,Cb のそれぞれの値が大きくなると該当する成分が明るくなり、小さくなると該当する成分が暗くなります。
また、Yb,Cr,Cb が 0 の場合は処理の前後で変化ありません。

処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
1,4,8 ビットイメージの場合は、**SelectMode** プロパティの値に関わらずイメージ全体に対して処理を行います。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

YCCRev(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの YCrCb 値の反転処理を行います。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Effect**->**YCCRev**(const TPoint * Points, const int Points_Size, bool Yb, bool Cr, bool Cb)
[*bool =*]*imagekitcontrolname*->**Effect**->**YCCRev**(bool Yb, bool Cr, bool Cb)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Effect**.**YCCRev**(const Points: array of TPoint; Yb, Cr, Cb: Boolean)
[*Boolean =*]*imagekitcontrolname*.**Effect**.**YCCRev**(Yb, Cr, Cb: Boolean)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。	
Yb	輝度 Y の反転(False: 反転なし, True: 反転あり)
Cr	色相 Cr の反転(False: 反転なし, True: 反転あり)
Cb	色相 Cb の反転(False: 反転なし, True: 反転あり)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Yb,Cr,Cb を True にするとそれぞれの YCrCb 値を反転します。
例えば、Yb が 255 の場合は 0 となります。(Yb = 255 - Yb)

処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。
SelectMode プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。
1,4,8 ビットイメージの場合は、**SelectMode** プロパティの値に関わらずイメージ全体に対して処理を行います。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

YCCSpline(イメージキットコントロール／Effect メソッド)

【機能】
ラスタイメージの YCrCb 値のスプライン補正を行います。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Effect**->**YCCSpline**(const TPoint * Points, const int Points_Size, const TPoint * spPoints, const int spPoints_Size, bool Yb, bool Cr, bool Cb)
[*bool* =]*imagekitcontrolname*->**Effect**->**YCCSpline**(const TPoint * spPoints, const int spPoints_Size, bool Yb, bool Cr, bool Cb)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Effect**.**YCCSpline**(const Points: array of TPoint; const spPoints: array of TPoint; Yb, Cr, Cb: Boolean)
[*Boolean* =]*imagekitcontrolname*.**Effect**.**YCCSpline**(const spPoints: array of TPoint; Yb, Cr, Cb: Boolean)

【引数】

名称	内容
Points	多角形の x,y 座標を指定する配列(座標はピクセル単位)
※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。	
spPoints	スプラインの通過 x,y 座標を指定する配列(座標は 0～255 の間) (値の設定方法は Points と同じ、配列の要素数は 3～10 が有効)
※C++Builder の場合、spPoints の要素数-1 を spPoints_Size に与えます。	
Yb	輝度 Y の成分に対して処理を行うかどうかの設定(False:しない True:する)
Cr	色相 Cr の成分に対して処理を行うかどうかの設定(False:しない True:する)
Cb	色相 Cb の成分に対して処理を行うかどうかの設定(False:しない True:する)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
入力されたイメージデータの YCrCb 値に対してスプライン曲線により変化させることができます。
使い方はイメージキットコントロールの RGBSpline メソッドの「使用例」を参照してください。

処理対象となるイメージハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(1,4,8,16,24,32 ビットイメージが対象で **LayerNo** プロパティによって処理されるイメージハンドルが決まります)。必要に応じてマスクイメージのハンドルを **MaskImageHandle** プロパティに設定します。
マスクハンドルを基に処理を行いたい場合は、**SelectMode** プロパティに vikEffectMask を **MaskImageHandle** プロパティに有効な値を設定します。イメージ全体に対して処理を行いたい場合は、**SelectMode** プロパティに vikEffectAll を設定します。選択した多角形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectPolygon を設定し、引数 Points に有効な値を設定します。その場合 Points の要素数は 2 以上でなければなりません。選択した円形に対して処理を行いたい場合には、**SelectMode** プロパティに vikEffectEllipse を設定し **RectLeft**,**RectTop**,**RectRight**,**RectBottom** プロパティに有効な値を設定します。**SelectMode** プロパティが vikEffectPolygon と vikEffectEllipse の場合には **InOut** プロパティに値の設定が必要です。**SelectMode** プロパティが vikEffectPolygon 以外の場合には Points にダミーの配列(もしくは NULL)を与えるか、もしくは Points が不要なメソッドをご利用ください。1,4,8 ビットイメージの場合は、**SelectMode** プロパティの値に関わらずイメージ全体に対して処理を行います。
成功した場合、処理後のラスタイメージのメモリハンドルは **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。

FileIO（イメージキットコントロール／カスタム階層プロパティ）

【機能】
イメージファイルの読込・保存機能を提供します。

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
ButtonName	イメージ読込・保存処理中のダイアログボックスに表示するボタンの名称を設定
Cancel	処理の中止の設定
Caption	イメージ読込・保存処理中のダイアログボックスに表示するタイトルバーの名称を設定
Comment	イメージファイルのコメント
CreationTimeDay	GetImageFileType メソッド実行後に取得される、ファイルの作成日時の日
CreationTimeHour	GetImageFileType メソッド実行後に取得される、ファイルの作成日時の時間
CreationTimeMinute	GetImageFileType メソッド実行後に取得される、ファイルの作成日時の分
CreationTimeMonth	GetImageFileType メソッド実行後に取得される、ファイルの作成日時の月
CreationTimeSecond	GetImageFileType メソッド実行後に取得される、ファイルの作成日時の秒
CreationTimeYear	GetImageFileType メソッド実行後に取得される、ファイルの作成日時の年
ExifAutoRotate	読み込み対象ファイルが Exif 形式の場合、主画像の方向に合わせて回転して読み込む
ExtendedDialog	ファイル選択ダイアログでのプレビューとファイル情報の設定
FileBitCount	GetImageFileType(Mem)メソッド実行後に取得される、ビット／ピクセル
FileExt	OpenFileDialog、SaveFileDialog メソッド実行時に使用するファイルタイプを設定
FileHeight	GetImageFileType(Mem)メソッド実行後に取得される、イメージの高さ（ピクセル）
FileImageSize	GetImageFileType(Mem)メソッド実行後に取得される、イメージのサイズ
FileMaxPage	GetImageFileType(Mem)メソッド実行後に取得される、イメージファイルのページの数
FileName	イメージ読込・保存時のファイル名を設定
FilePath	OpenFileDialog、SaveFileDialog メソッド実行時に使用するフォルダを設定
FileSize	GetImageFileType(Mem)メソッド実行後に取得される、イメージファイルのサイズ（バイト）
FileType	GetImageFileType(Mem)メソッド実行後に取得される、ファイルタイプ
FileWidth	GetImageFileType(Mem)メソッド実行後に取得される、イメージの幅（ピクセル）
FileWidthByte	GetImageFileType(Mem)メソッド実行後に取得される、イメージの 1 ラインのバイト数
FileXdpi	GetImageFileType(Mem)メソッド実行後に取得される、イメージの横方向の DPI
FileYdpi	GetImageFileType(Mem)メソッド実行後に取得される、イメージの縦方向の DPI
GifAnime	マルチイメージ GIF かどうかの設定
GifAnimeDelay	マルチイメージ GIF のイメージ間の表示する時間の設定
ImageHandleRawData	Raw データを示すメモリハンドル
ImageHandleRawDataSize	Raw データを示すメモリハンドルが使用しているサイズ
Information	ファイル選択ダイアログでのファイル情報の設定
Interlace	GIF/PNG 形式のインタレースの設定
JPEG2000CodeBlockHeight	JPEG2000 形式の縦方向のコードブロックサイズ
JPEG2000CodeBlockWidth	JPEG2000 形式の横方向のコードブロックサイズ
JPEG2000NumResLevel	JPEG2000 形式の解像度レベル
JPEG2000PrecinctHeight	JPEG2000 形式の縦方向のプレシンクトサイズ
JPEG2000PrecinctWidth	JPEG2000 形式の横方向のプレシンクトサイズ
JPEG2000Reversible	JPEG2000 形式の圧縮方法を設定
JPEG2000Size	JPEG2000 形式のファイルの保存サイズを設定
JPEG2000TileHeight	JPEG2000 形式の縦方向のタイルサイズ
JPEG2000TileWidth	JPEG2000 形式の横方向のタイルサイズ
JpegQuality	イメージ保存時の JPEG 圧縮品質係数を設定
JpegSubsamp	YCrCb プレーンのサブサンプリング
LastAccessTimeDay	GetImageFileType メソッド実行後に取得される、ファイルのアクセス日時の日
LastAccessTimeHour	GetImageFileType メソッド実行後に取得される、ファイルのアクセス日時の時間
LastAccessTimeMinute	GetImageFileType メソッド実行後に取得される、ファイルのアクセス日時の分
LastAccessTimeMonth	GetImageFileType メソッド実行後に取得される、ファイルのアクセス日時の月
LastAccessTimeSecond	GetImageFileType メソッド実行後に取得される、ファイルのアクセス日時の秒
LastAccessTimeYear	GetImageFileType メソッド実行後に取得される、ファイルのアクセス日時の年
LastWriteTimeDay	GetImageFileType メソッド実行後に取得される、ファイルの更新日時の日

LastWriteTimeHour	GetImageFileType メソッド実行後に取得される、ファイルの更新日時の時間
LastWriteTimeMinute	GetImageFileType メソッド実行後に取得される、ファイルの更新日時の分
LastWriteTimeMonth	GetImageFileType メソッド実行後に取得される、ファイルの更新日時の月
LastWriteTimeSecond	GetImageFileType メソッド実行後に取得される、ファイルの更新日時の秒
LastWriteTimeYear	GetImageFileType メソッド実行後に取得される、ファイルの更新日時の年
LoadPage	FPX/GIF/TIFF 形式のマルチイメージ用の読込ページの設定
Message	イメージ読込・保存処理中のダイアログボックスに表示するメッセージを設定
PalBlue	透過パレットのカラー値(青)の設定
PalGreen	透過パレットのカラー値(緑)の設定
PalRed	透過パレットのカラー値(赤)の設定
PngAlphaChannel	PNG のアルファチャンネルの設定
Preview	ファイル選択ダイアログでのプレビュー設定
SaveFileDlgFileType	SaveFileDialog メソッドで選択されたファイルの種類
TiffAppend	TIFF 形式の追加保存の設定
TiffColorSpace	TIFF 形式のフルカラーモードの設定
TiffSaveOneStrip	TIFF 形式のストリップの設定
Transparent	GIF/PNG 形式の透過の設定
VectorHeight	ベクトルイメージ読み込み時の高さ(ピクセル)
VectorWidth	ベクトルイメージ読み込み時の幅(ピクセル)

【ImageKit7/8/9/10 ActiveX との違い】
変更されたプロパティ: JPEG2000NumrResLevel --> JPEG2000NumResLevel

●メソッド一覧(アルファベット順)

カスタムメソッド	内容
CMYKBmpPlaneFileLoad	CMYK プレーン毎に保存してある BMP ファイルをロード
CMYKBmpPlaneFileSave	ラスタイメージを CMYK プレーン毎に BMP 形式でファイルへ保存
FileLoadAsRawData	イメージファイルから Raw データをメモリ上にロード
FileSaveAsRawData	Raw データをファイルへ保存
FTPConnect	FTP サーバへの接続
FTPDeleteFile	FTP サーバに存在するファイルを削除(接続と切断も行う)
FTPDeleteFileEx	FTP サーバに存在するファイルを削除
FTPDisconnect	接続している FTP サーバの切断
FTPGetFile	FTP サーバからファイルを取得(接続と切断も行う)
FTPGetFileEx	FTP サーバからファイルを取得
FTPPutFile	FTP サーバへファイルを転送(接続と切断も行う)
FTPPutFileEx	FTP サーバへファイルを転送
FTPRenameFile	FTP サーバに存在するファイル名を変更(接続と切断も行う)
FTPRenameFileEx	FTP サーバに存在するファイル名を変更
GetImageFileType	ファイル名からファイルタイプの取得
GetImageFileTypeMem	Raw データからファイルタイプの取得
HTTPConnect	HTTP(S)サーバへの接続
HTTPDisconnect	接続している HTTP(S)サーバの切断
HTTPGetFile	HTTP サーバからファイルを取得(接続と切断も行う)
HTTPGetFileEx	HTTP(S)サーバからファイルを取得
HTTPPutFile	HTTP サーバへファイルを転送(接続と切断も行う)
HTTPPutFileEx	HTTP(S)サーバへファイルを転送
HTTPSGetFile	HTTPS サーバからファイルを取得(接続と切断も行う)
HTTPSPutFile	HTTPS サーバへファイルを転送(接続と切断も行う)
LoadFile	ファイルからの読み込み
LoadFileMem	Raw データからの読み込み
LoadFromStream	ストリームからの読み込み
OpenFileDialog	プレビュー付きファイル選択ダイアログ(オープン)
RGBBmpPlaneFileLoad	RGB プレーン毎に保存してある BMP ファイルをロード
RGBBmpPlaneFileSave	ラスタイメージを RGB プレーン毎に BMP 形式でファイルへ保存
SaveFile	ファイルへ保存
SaveFileDialog	プレビュー付きファイル選択ダイアログ(セーブ)
SaveFileMem	Raw データへ保存

SaveToStream	ストリームへ保存
YCCBmpPlaneFileLoad	YCrCb プレーン毎に保存してある BMP ファイルをロード
YCCBmpPlaneFileSave	ラスタイメージを YCrCb プレーン毎に BMP 形式でファイルへ保存

●階層プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Exif	Exif(JPEG)ファイルから主画像とサムネイル画像の情報を取得
PDF	PDF 形式への保存を行う

※ImageKit10 で対応している GIF 形式は、
保存時： GIF89a
読込時： GIF87a、GIF89a
です。

ButtonName,Caption,Message(イメージキットコントロール／FileIO プロパティ)

【機能】
イメージ読込・保存処理中のダイアログボックスに表示するボタン、キャプション、メッセージを取得または設定します。

【書式】
※**ButtonName** にて説明(**Caption**,**Message** も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->**FileIO**->**ButtonName** [= *UnicodeString*]
(2)Delphi　　*imagekitcontrolname*.**FileIO**.**ButtonName** [= *string*]

【設定値】
ButtonName プロパティは、処理中のダイアログボックスのボタンに表示する文字列。
Caption プロパティは、処理中のダイアログボックスのタイトルバーに表示する文字列。
Message プロパティは、処理中のダイアログボックスの中央に表示する文字列。

【解説】
ButtonName,**Caption**,**Message** プロパティ全てに何も設定しなかった場合は、進捗状況のダイアログボックスは表示されず、**Progress** イベントが発生します。
逆にどれか一つでも有効な文字列を設定した場合は、進捗状況ダイアログボックスが表示され、**Progress** イベントは発生しません。

【値の設定】 実行時

【値の参照】 実行時

Cancel(イメージキットコントロール／FileIO プロパティ)

【機能】
イメージの読込・保存処理を中止するかどうかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Cancel** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**Cancel** [= *Boolean*]

【設定値】

値	説明
True	読込・保存処理を中止する
False	読込・保存処理を中止しない

【解説】
Cancel プロパティに True を設定すると、読込・保存処理を中止できます。

【値の設定】 実行時

【値の参照】 不可

Comment(イメージキットコントロール／FileIO プロパティ)

【機能】
イメージファイルに保存するコメントを設定したり、GetImageFileType(Mem)メソッドを実行して取得したコメントを表します。

【書式】
(1)C++Builder　*imagekitcontrolname*->FileIO->Comment [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.Comment [= *string*]

【設定値】
コメントの文字列。(MAX:2047 バイト)

【解説】
コメントはファイル形式により異なります。
以下に説明のないファイル形式はコメントをサポートしておりませんので、設定しても無視されます。

FPX	タイトル＋0x0d＋サブタイトル＋0x0d＋作成者＋0x0d＋コメント＋0x0d
GIF	コメント
JPEG	コメント
JPEG2000	タイトル＋0x0d＋サブタイトル＋0x0d＋作成者＋0x0d＋コメント＋0x0d
PNG	タイトル＋0x0d＋作成者＋0x0d＋説明＋0x0d＋版権＋0x0d＋時間＋0x0d＋ソフトウェア＋0x0d＋権利放棄＋0x0d＋警告＋0x0d＋入力機器＋0x0d＋コメント＋0x0d
TIFF	文書名＋0x0d＋画像説明＋0x0d＋ページ名＋0x0d＋ソフトウェア＋0x0d＋日時＋0x0d＋作者＋0x0d

必要のない項目については空の文字列を設定してください。
※16 進表記のため、Delphi は 0x を$に置き換えてください。

【値の設定】 実行時

【値の参照】 実行時

CreationTimeDay,CreationTimeHour,CreationTimeMinute,CreationTimeMonth, CreationTimeSecond,CreationTimeYear,LastAccessTimeDay,LastAccessTimeHour, LastAccessTimeMinute,LastAccessTimeMonth,LastAccessTimeSecond,LastAccessTimeYear, LastWriteTimeDay,LastWriteTimeHour,LastWriteTimeMinute,LastWriteTimeMonth, LastWriteTimeSecond,LastWriteTimeYear(イメージキットコントロール／FileIO プロパティ)

【機能】
GetImageFileType メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※CreationTimeDay にて説明(他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->CreationTimeDay [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.CreationTimeDay [= *Smallint*]

【参照値】
CreationTimeDay はファイルの作成日時の日を表します。
CreationTimeHour はファイルの作成日時の時間を表します。
CreationTimeMinute はファイルの作成日時の分を表します。
CreationTimeMonth はファイルの作成日時の月を表します。
CreationTimeSecond はファイルの作成日時の秒を表します。
CreationTimeYear はファイルの作成日時の年を表します。

LastAccessTimeDay はファイルのアクセス日時の日を表します。
LastAccessTimeHour はファイルのアクセス日時の時間を表します。
LastAccessTimeMinute はファイルのアクセス日時の分を表します。
LastAccessTimeMonth はファイルのアクセス日時の月を表します。
LastAccessTimeSecond はファイルのアクセス日時の秒を表します。
LastAccessTimeYear はファイルのアクセス日時の年を表します。

LastWriteTimeDay はファイルの更新日時の日を表します。
LastWriteTimeHour はファイルの更新日時の時間を表します。
LastWriteTimeMinute はファイルの更新日時の分を表します。
LastWriteTimeMonth はファイルの更新日時の月を表します。
LastWriteTimeSecond はファイルの更新日時の秒を表します。
LastWriteTimeYear はファイルの更新日時の年を表します。

【解説】
ファイルの作成日時・アクセス日時・更新日時を表します。
GetImageFileTypeMem メソッドを実行すると各プロパティに 0 が設定されます。

【値の設定】 不可

【値の参照】 実行時

ExifAutoRotate(イメージキットコントロール／FileIO プロパティ)

【機能】
読み込み対象ファイルが Exif 形式の場合、主画像の方向に合わせて回転を行って読み込みます。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->ExifAutoRotate** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.ExifAutoRotate** [= *Boolean*]

【設定値】

値	説明
True	主画像の方向に合わせて回転して読み込む
False	主画像をありのまま読み込む(方向を無視)

【解説】
デフォルトは True です。当プロパティの値は **LoadFile,LoadFileMem** メソッドで使用されますが、読み込み対象ファイルが Exif 形式以外の場合は無効です。

【値の設定】 実行時

【値の参照】 不可

ExtendedDialog(イメージキットコントロール／FileIO プロパティ)

【機能】
ファイル選択ダイアログのプレビューとファイル情報の表示可否を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->ExtendedDialog** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.ExtendedDialog** [= *Boolean*]

【設定値】
ファイル選択ダイアログの初期表示時のモード。

値	説明
True	プレビューとファイル情報のチェックボックスを表示
False	プレビューとファイル情報のチェックボックスを非表示

【解説】
デフォルトは True です。

【値の設定】 実行時

【値の参照】 実行時

FileBitCount,FileHeight,FileImageSize,FileMaxPage,FileSize, FileType,FileWidth,FileWidthByte,FileXdpi,FileYdpi （イメージキットコントロール／FileIO プロパティ）

【機能】

GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】

※FileBitCount にて説明（FileMaxPage,FileType,FileXdpi,FileYdpi も同様な使い方）

(1)C++Builder　*imagekitcontrolname*->FileIO->FileBitCount [= *short*]

(2)Delphi　*imagekitcontrolname*.FileIO.FileBitCount [= *Smallint*]

※FileHeight にて説明（FileImageSize,FileSize,FileWidth,FileWidthByte も同様な使い方）

(1)C++Builder　*imagekitcontrolname*->FileIO->FileHeight [= *int*]

(2)Delphi　*imagekitcontrolname*.FileIO.FileHeight [= *Integer*]

【参照値】

FileBitCount はイメージの 1 ピクセルあたりのビット数（1、4、8、16、24、32）。
FileHeight はイメージの縦のピクセル数。
FileImageSize はイメージサイズのバイト数。
FileMaxPage はイメージファイルのページ数。
FileSize はイメージファイルのサイズ。
FileType はイメージのファイルの種類（1～29）。
FileWidth はイメージの横のピクセル数。
FileWidthByte はイメージの 1 ラインのバイト数。
FileXdpi はイメージの横方向の 1 インチあたりのピクセル数。
FileYdpi はイメージの縦方向の 1 インチあたりのピクセル数。

【解説】

FileBitCount の示すイメージの色は

1:2 値	4:16 色	8:256 色
16:16 ﾋﾞｯﾄｶﾗｰ	24:24 ﾋﾞｯﾄｶﾗｰ	32:32 ﾋﾞｯﾄｶﾗｰ

となります。

FileType の示すファイルの種類は

1:BMP/DIB	2:BMP/DIB(RLE4)	3:BMP/DIB(RLE8)	4:JPEG(基本 DCT)
5:JPEG(ﾌﾟﾛｸﾞﾚｯｼﾌﾞ DCT)	6:GIF(87a)	7:GIF(89a)	8:TIFF(非圧縮)
9:TIFF(CCITTRLE)	10:TIFF(GROUP3-1D)	11:TIFF(GROUP3-2D)	12:TIFF(GROUP4)
13:TIFF(PACKBITS)	14:TIFF(LZW)	15:PNG	16:FPX(非圧縮)
17:FPX(Single Color)	18:FPX(JPEG)	19:PCX	20:WMF
21:EMF	22:DXF	23:Exif(JPEG)	24:SVG
25:JPEG2000(Part1)	26:JPEG2000(Code Stream)	27:SXF(p21)	28:SFC(sfc)
29:TIFF(JPEG)			

となります。
TIFF の GROUP3-1D は MH、GROUP3-2D は MR、GROUP4 は MMR と同じ形式です。
Exif はバージョン 2.3 の一部のタグにも対応しています。0 の場合は未対応の形式です。

定数(vikFileTypeBMP = 1, vikFileTypeBMPRLE4 = 2, vikFileTypeBMPRLE8 = 3, vikFileTypeJPEG = 4, vikFileTypeJPEGProgress = 5, vikFileTypeGIF87 = 6, vikFileTypeGIF89 = 7, vikFileTypeTIFFNoncompressed = 8, vikFileTypeTIFFCcittrle = 9, vikFileTypeTIFFGroup3_1D = 10, vikFileTypeTIFFGroup3_2D = 11, vikFileTypeTIFFGroup4 = 12, vikFileTypeTIFFPackbits = 13, vikFileTypeTIFFLZW = 14, vikFileTypePNG = 15, vikFileTypeFPXNoncompressed = 16, vikFileTypeFPXSingleColor = 17, vikFileTypeFPXJpeg = 18, vikFileTypePCX = 19, vikFileTypeWMF = 20, vikFileTypeEMF = 21, vikFileTypeDXF = 22, vikFileTypeEXIF = 23, vikFileTypeSVG = 24, vikFileTypeJPEG2000 = 25, vikFileTypeJPEG2000Stream = 26, vikFileTypeSXFP21 = 27, vikFileTypeSXFSFC = 28, vikFileTypeTIFFJPEG = 29)を使用することも可能です。

ベクトルイメージでは、FileBitCount,FileImageSize,FileMaxPage,FileWidthByte プロパティは使用しません。
対応しているファイル形式であっても、ファイルに該当する情報が保存されていない場合はプロパティに 0 が設定されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
FileType プロパティの型が 4 バイト型ではなく 2 バイト型です。

FileExt,FilePath(イメージキットコントロール／FileIO プロパティ)

【機能】
OpenFileDialog メソッドや SaveFileDialog メソッドで初期表示するフォルダやファイルのタイプを取得または設定します。

【書式】
※FileExt にて説明(FilePath も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->FileIO->FileExt [= *UnicodeString*]
(2)Delphi　　*imagekitcontrolname*.FileIO.FileExt [= *string*]

【設定値】
FileExt はファイルのタイプ。
FilePath はフォルダ名。

【解説】
FileExt に複数のタイプを指定する場合には、ファイルの拡張子をセミコロン(;)で区切ります。
"*"を設定すると、種類のところに「全てのファイル」は表示されません。
拡張子を"<JPG>"のように<>で囲むとデフォルトとして表示できます。
また、アルファベットの大文字と小文字は区別しません。("BMP"と"bmp"を同じとみなします)

例
(1)種類に「全てのファイル」、「BMP」、「JPG」、「PNG」の 4 つが設定される。
VImageKit1.FileIO.FileExt := 'BMP;JPG;PNG;';
(2)種類に「BMP」、「JPG」、「PNG」の 3 つが設定される。
VImageKit1.FileIO.FileExt := '*;BMP;JPG;PNG;';

【値の設定】 実行時

【値の参照】 実行時

FileName（イメージキットコントロール／FileIO プロパティ）

【機能】
イメージの読込・保存時のファイル名を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**FileName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**FileName** [= *string*]

【設定値】
ファイル名（フルパス）

【解説】
(1)ローカルドライブやネットワークドライブのファイルを読み込む場合

```
VImageKit1.FileIO.FileName := 'c:¥abc.jpg';
VImageKit1.FileIO.LoadFile(vikLoad);
```

(2)FTP サーバーに配置されているファイルを読み込む場合

```
VImageKit1.FileIO.FileName := 'ftp://www.newtone.co.jp/image/abc.jpg;;;true;user;password'; //(*)
VImageKit1.FileIO.LoadFile(vikLoad);
```

(*)値を設定する際の順番
FTP://サーバ名/フォルダ名/ファイル名;プロキシ名;ポート番号;パッシブモード;ユーザ名;パスワード

プロキシサーバを設定したり、ポート番号やパッシブ（PASV）モード接続を変更する場合はプロパティに「FTP サーバの名称もしくは IP アドレス;プロキシサーバの名称または IP アドレス;ポート番号;パッシブモード接続の有無」を渡してください。（文字列をセミコロンで区切る）
なお、ポート番号を省略するとデフォルトポートが割り当てられ、パッシブモード接続の有無（"true"または"false" - 大小文字関係なし)を省略するとパッシブモード接続となります。

(3)HTTP(S)サーバーに配置されているファイルを読み込む場合

```
VImageKit1.FileIO.FileName := 'http://www.newtone.co.jp/image/abc.jpg;;;;user;password'; //(**)
VImageKit1.FileIO.LoadFile(vikLoad);
```

(**)値を設定する際の順番
HTTP://サーバ名/フォルダ名/ファイル名;プロキシ名;ポート番号;HTTPS;ユーザ名;パスワード

プロキシサーバを設定したりポート番号を変更、または HTTPS を利用する場合はプロパティに「HTTP(S)サーバの名称もしくは IP アドレス;プロキシサーバの名称または IP アドレス;ポート番号;HTTPS」を渡してください。（文字列をセミコロンで区切る）
なお、ポート番号を省略するとデフォルトポートが割り当てられ、"HTTPS"を省略すると HTTP サーバからの処理となります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
ローカルドライブやネットワークドライブのファイルだけではなく、FTP サーバーや HTTP(S)サーバーに配置されているファイルも設定できるようになりました。

GifAnime(イメージキットコントロール／FileIO プロパティ)

【機能】
マルチイメージ GIF かどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**GifAnime** [*= bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**GifAnime** [*= Boolean*]

【設定値】

値	説明
True	アニメーション(マルチイメージ)
False	シングルイメージ

【解説】
イメージを GIF ファイルに保存する際に設定します。デフォルトは False です。

【値の設定】 実行時

【値の参照】 実行時

GifAnimeDelay(イメージキットコントロール／FileIO プロパティ)

【機能】
マルチイメージ GIF のイメージ間の表示する時間を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->GifAnimeDelay** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.GifAnimeDelay** [= *Smallint*]

【設定値】
画像を表示した後、次の画像を表示するまでの時間(1/100 秒単位)。

【解説】
イメージを GIF ファイルに保存する際に設定します。デフォルトは 0 です。
ただし、**GifAnime** プロパティが True の場合のみ有効になります。

【値の設定】 実行時

【値の参照】 実行時

ImageHandleRawData(イメージキットコントロール／FileIO プロパティ)

【機能】
イメージ読込・保存用の Raw データのメモリハンドルを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**ImageHandleRawData** [= *NativeUInt*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**ImageHandleRawData** [= *THandle*]

【設定値】
読込・保存の対象となる Raw データのメモリハンドル。

【解説】
FileLoadAsRawData メソッドを実行するとプロパティに Raw データが設定され、**FileSaveAsRawData** メソッドを実行するとプロパティに設定された Raw データがファイルに保存されます。

【値の設定】 実行時

【値の参照】 実行時

ImageHandleRawDataSize（イメージキットコントロール／FileIO プロパティ）

【機能】
ImageHandleRawData プロパティに設定されている Raw データのメモリサイズを取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**ImageHandleRawDataSize** [= *int*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**ImageHandleRawDataSize** [= *Integer*]

【参照値】
Raw データが使用しているメモリサイズ（単位はバイト）。

【値の設定】 不可

【値の参照】 実行時

Information(イメージキットコントロール／FileIO プロパティ)

【機能】
ファイル選択ダイアログでファイル情報の表示可否を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Information** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**Information** [= *Boolean*]

【設定値】
ファイル選択ダイアログの初期表示時のモード。

値	説明
True	ファイル情報を表示(チェックボックスにチェックが入った状態)
False	ファイル情報を非表示(チェックボックスにチェックが入っていない状態)

【解説】
デフォルトは False です。
OpenFileDialog メソッドや **SaveFileDialog** メソッドを実行した後でも、**ExtendedDialog** プロパティが True であればプロパティの値に関係なくファイル情報の表示は可能です。

【値の設定】 実行時

【値の参照】 実行時

Interlace（イメージキットコントロール／FileIO プロパティ）

【機能】
イメージ保存時（GIF/PNG 形式）のインタレースを指定したり、GetImageFileType(Mem)メソッドを実行して取得した値を表します。

【書式】
(1)C++Builder　*imagekitcontrolname*->FileIO->Interlace [= *bool*]
(2)Delphi　*imagekitcontrolname*.FileIO.Interlace [= *Boolean*]

【設定値】

値	説明
True	インタレース
False	ノンインタレース

【解説】
イメージを GIF または PNG ファイルに保存する際に設定します。デフォルトは False です。

【値の設定】 実行時

【値の参照】 実行時

JPEG2000CodeBlockHeight,JPEG2000CodeBlockWidth(イメージキットコントロール／FileIO プロパティ)

【機能】
JPEG2000 形式のコードブロックサイズを取得または設定します。

【書式】
※JPEG2000CodeBlockHeight にて説明(JPEG2000CodeBlockWidth も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->JPEG2000CodeBlockHeight [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.JPEG2000CodeBlockHeight [= *Smallint*]

【設定値】
JPEG2000CodeBlockHeigh は縦方向のコードブロックサイズ。
JPEG2000CodeBlockWidth は横方向のコードブロックサイズ。

【解説】
デフォルト値は 0 です。コードブロックサイズには 2 のべき乗"2～10"を指定します(2^2～2^10)。
プロパティ値は JPEG2000 形式で保存する際に使用されます。

【値の設定】 実行時

【値の参照】 実行時

JPEG2000NumResLevel(イメージキットコントロール／FileIO プロパティ)

【機能】
JPEG2000 形式の解像度レベルを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->JPEG2000NumResLevel** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.JPEG2000NumResLevel** [= *Smallint*]

【設定値】
解像度レベル(0～)。

【解説】
デフォルト値は 0 です。
プロパティ値は JPEG2000 形式で保存する際に使用されます。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
プロパティの名称が **JPEG2000NumrResLevel** から変更されました。

JPEG2000PrecinctHeight,JPEG2000PrecinctWidth（イメージキットコントロール／FileIO プロパティ）

【機能】
JPEG2000 形式のプレシンクトサイズを取得または設定します。

【書式】
※JPEG2000PrecinctHeight にて説明（JPEG2000PrecinctWidth も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->FileIO->JPEG2000PrecinctHeight [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.JPEG2000PrecinctHeight [= *Smallint*]

【設定値】
JPEG2000PrecinctHeigh は縦方向のプレシンクトサイズ。
JPEG2000PrecinctWidth は横方向のプレシンクトサイズ。

【解説】
デフォルト値は 0 です。プレシンクトサイズには 2 のべき乗"1～15"を指定します（2^1～2^15）。
プロパティ値は JPEG2000 形式で保存する際に使用されます。

【値の設定】 実行時

【値の参照】 実行時

JPEG2000Reversible(イメージキットコントロール／FileIO プロパティ)

【機能】
JPEG2000 形式の圧縮方法を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->JPEG2000Reversible** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.JPEG2000Reversible** [= *Boolean*]

【設定値】

値	説明
True	可逆圧縮
False	非可逆圧縮

【解説】
デフォルトは True です。
True の場合、ファイルサイズは大きくなりますが画像の劣化はありません。False の場合、**JPEG2000Size** プロパティにファイルの保存サイズを設定します。
プロパティ値は JPEG2000 形式で保存する際に使用されます。

【値の設定】 実行時

【値の参照】 実行時

JPEG2000Size(イメージキットコントロール／FileIO プロパティ)

【機能】
JPEG2000 形式のファイルの保存サイズを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**JPEG2000Size** [= *double*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**JPEG2000Size** [= *Double*]

【設定値】
非圧縮時のイメージサイズに対する倍数。

【解説】
プロパティ値が 0.01 であれば非圧縮時に比べてファイルサイズが 1/100 となります。デフォルト値は 0.1 です。
JPEG2000Reversible プロパティが False の場合に有効です。プロパティ値は JPEG2000 形式で保存する際に使用されます。

【値の設定】 実行時

【値の参照】 実行時

JPEG2000TileHeight,JPEG2000TileWidth(イメージキットコントロール／FileIO プロパティ)

【機能】
JPEG2000 形式のタイルサイズを取得または設定します。

【書式】
※JPEG2000TileHeight にて説明(JPEG2000TileWidth も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->JPEG2000TileHeight [= *int*]
(2)Delphi　*imagekitcontrolname*.FileIO.JPEG2000TileHeight [= *Integer*]

【設定値】
JPEG2000TileHeight は縦方向のタイルサイズ(ピクセル単位)。
JPEG2000TileWidth は横方向のタイルサイズ(ピクセル単位)。

【解説】
デフォルト値は 0 です。
プロパティ値は JPEG2000 形式で保存する際に使用されます。

【値の設定】 実行時

【値の参照】 実行時

JpegQuality(イメージキットコントロール／FileIO プロパティ)

【機能】
イメージ保存時(JPEG 形式または FPX の JPEG 形式)の圧縮品質係数を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**JpegQuality** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**JpegQuality** [= *Smallint*]

【設定値】
0～100(推奨 75)。

【解説】
デフォルト値は 75 です。
この値が小さいほどファイルサイズは小さくなりますが、イメージの品質は悪くなります。逆に、この値が大きいほどファイルサイズは大きくなりますが、イメージの品質は良くなります。
イメージを JPEG または FPX(JPEG 形式)ファイルに保存する際に設定します。

【値の設定】 実行時

【値の参照】 実行時

JpegSubsamp(イメージキットコントロール／FileIO プロパティ)

【機能】
イメージ保存時(JPEG 形式)の YCrCb のサブサンプリングを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**JpegSubsamp** [= *TVIkJpegSubsamp*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**JpegSubsamp** [= *TVIkJpegSubsamp*]

【TVIkJpegSubsamp 型】
ユニット
IkInit
type
TVIkJpegSubsamp = (vik411, vik422, vik444);

【設定値】

値	説明
vik411	Y:Cr:Cb　4:1:1
vik422	Y:Cr:Cb　4:2:2
vik444	Y:Cr:Cb　4:4:4

【解説】
Y は輝度情報、Cr と Cb は色相情報を表します。保存するイメージが 24 ビットカラーの時に有効になります。
イメージを JPEG ファイルに保存する際に設定します。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ik411, ik422, ik444)。

LoadPage(イメージキットコントロール／FileIO プロパティ)

【機能】
複数イメージ読込時(FPX/GIF/TIFF 形式)の読込むページを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**LoadPage** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**LoadPage** [= *Smallint*]

【設定値】
読込むイメージのページ数。
最初のページは 0 です。(FPX の基画像も 0 です。)

【解説】
GIF/TIFF 形式では、複数のイメージをひとつのファイルに格納してある「マルチイメージファイル」が存在する場合があります。またFPX 形式は、マルチ・レゾリューション(複数解像度)で基画像から最小画像までの画素数の異なる複数画面を1つのファイルに持つことができます。
LoadPage プロパティは、その「マルチイメージファイル」から指定した 1 イメージ(ページ)をメモリ上に読み込む時に使用します。**LoadPage** プロパティに設定したページがない場合は、**LoadFile** メソッドもしくは **LoadFileMem** メソッド実行後、False が返されます。

【値の設定】 実行時

【値の参照】 実行時

PalBlue,PalGreen,PalRed(イメージキットコントロール／FileIO プロパティ)

【機能】

イメージを透明 GIF/PNG でファイルに保存する場合の、透明色に使う赤・緑・青(RGB)の各カラー値を取得または設定します。

【書式】

※**PalBlue** にて説明(**PalGreen**、**PalRed** も同様な使い方)

(1)C++Builder　　*imagekitcontrolname*->**FileIO**->**PalBlue** [= *short*]

(2)Delphi　　*imagekitcontrolname*.**FileIO**.**PalBlue** [= *Smallint*]

【参照値】

0～255。

【解説】

透明 GIF/PNG 形式でのファイル保存で透明色を設定する場合は、この **PalBlue,PalGreen,PalRed** の各プロパティに該当する青・緑・赤の各カラー値をセットします。

その際、**PalBlue,PalGreen,PalRed** の各プロパティに直接各カラー値をセットしても構いませんが、イメージキットコントロール直下の **PalBlue,PalGreen,PalRed** の各プロパティの各カラー値をそのままセットすることで、簡単に実際のイメージを見ながら決定した各カラー値を透明色とすることができます。

Transparent プロパティが True の時にそれぞれに設定した値が有効となります。

イメージを GIF または PNG ファイルに保存する際に設定します。

【値の設定】 実行時

【値の参照】 実行時

PngAlphaChannel(イメージキットコントロール／FileIO プロパティ)

【機能】
PNG 形式のイメージの読み込み、および保存時のアルファチャンネルを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->PngAlphaChannel** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PngAlphaChannel** [= *Boolean*]

【設定値】

値	説明
True	アルファチャンネルを独立したプレーンとする(RGBA 形式)
False	アルファチャンネルを考慮する(RGB 形式)

【解説】
デフォルトは False です。

【値の設定】 実行時

【値の参照】 実行時

Preview(イメージキットコントロール／FileIO プロパティ)

【機能】
ファイル選択ダイアログのプレビュー表示の可否を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Preview** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**Preview** [= *Boolean*]

【設定値】
ファイル選択ダイアログの初期表示時のモード。

値	説明
True	プレビュー表示あり(チェックボックスにチェックが入った状態)
False	プレビュー表示なし(チェックボックスにチェックが入っていない状態)

【解説】
デフォルトは False です。
OpenFileDialog メソッドや **SaveFileDialog** メソッドを実行した後でも、**ExtendedDialog** プロパティが True であればプロパティの値に関係なくプレビュー表示は可能です。

【値の設定】 実行時

【値の参照】 実行時

SaveFileDlgFileType(イメージキットコントロール／FileIO プロパティ)

【機能】
SaveFileDialog メソッドで選択されたファイルの種類を表します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->SaveFileDlgFileType** [*= short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.SaveFileDlgFileType** [*= Smallint*]

【設定値】

値	説明
1	BMP
2	JPEG
3	GIF
4	TIFF
5	PNG
6	FPX
7	PCX
8	WMF
9	EMF
10	DXF
11	SVG
12	JPEG2000(Part1)
13	JPEG2000(Code Stream)
14	SXF(p21)
15	SXF(sfc)

定数(vikSaveDlgBMP = 1, vikSaveDlgJPEG = 2, vikSaveDlgGIF = 3, vikSaveDlgTIFF = 4, vikSaveDlgPNG = 5, vikSaveDlgFPX = 6, vikSaveDlgPCX = 7, vikSaveDlgWMF = 8, vikSaveDlgEMF = 9, vikSaveDlgDXF = 10, vikSaveDlgSVG = 11, vikSaveDlgJPEG2000 = 12, vikSaveDlgJPEG2000Stream = 13, vikSaveDlgSXFP21 = 14, vikSaveDlgSXFSFC = 15)を使用することも可能です。

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
・定数を使用する場合、識別子の先頭に v が付加されました(ActiveX は ikSaveDlgBMP, ikSaveDlgJPEG, ikSaveDlgGIF, ikSaveDlgTIFF, ikSaveDlgPNG, ikSaveDlgFPX, ikSaveDlgPCX, ikSaveDlgWMF, ikSaveDlgEMF, ikSaveDlgDXF, ikSaveDlgSVG, ikSaveDlgJPEG2000, ikSaveDlgJPEG2000Stream, ikSaveDlgSXFP21, ikSaveDlgSXFSFC)。
・プロパティの型が 4 バイト型ではなく 2 バイト型です。

TiffAppend(イメージキットコントロール／FileIO プロパティ)

【機能】
イメージ保存時(TIFF 形式)の保存するモードを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->TiffAppend** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.TiffAppend** [= *Boolean*]

【設定値】

値	説明
True	マルチイメージ形式(既存のファイルの一番最後に保存)
False	シングルイメージ形式(1 ファイル 1 イメージ)

【解説】
イメージを TIFF ファイルに保存する際に設定します。デフォルトは False です。

【値の設定】 実行時

【値の参照】 実行時

TiffColorSpace(イメージキットコントロール／FileIO プロパティ)

【機能】
イメージ保存時(TIFF 形式)のフルカラーモードを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->TiffColorSpace** [= *TVIkTiffColorSpace*]
(2)Delphi　*imagekitcontrolname*.**FileIO.TiffColorSpace** [= *TVIkTiffColorSpace*]

【TVIkTiffColorSpace 型】
```
ユニット
  IkInit
type
  TVIkTiffColorSpace = (vikRGB, vikCMYK);
```

【設定値】

値	説明
vikRGB	RGB
vikCMYK	CMYK

【解説】
保存するイメージが 24,32 ビットカラーの時に有効になります。デフォルトは vikRGB です。
vikCMYK の場合は、出力されるファイルのビット数は 32 となります。
イメージを TIFF(非圧縮、PACKBITS、LZW)ファイルに保存する際に設定します。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikRGB, ikCMYK)。

TiffSaveOneStrip（イメージキットコントロール／FileIO プロパティ）

【機能】
イメージ保存時（TIFF 形式）のストリップを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->TiffSaveOneStrip** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.TiffSaveOneStrip** [= *Boolean*]

【設定値】

値	説明
True	1 つのストリップで保存
False	複数のストリップで保存

【解説】
デフォルトは False です。プロパティの設定が有効になるのは、TIFF の GROUP4 形式を除く場合です。（GROUP4 は 1 ストリップ固定です）

【値の設定】 実行時

【値の参照】 実行時

Transparent（イメージキットコントロール／FileIO プロパティ）

【機能】
GIF/PNG 形式の透過を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Transparent** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Transparent** [= *Boolean*]

【設定値】

値	説明
True	透過有効
False	透過無効

【解説】
イメージを GIF または PNG ファイルに保存する際に設定します。デフォルトは False です。
また、透明色は **PalBlue,PalGreen,PalRed** の各プロパティで設定します。

【値の設定】 実行時

【値の参照】 実行時

VectorHeight,VectorWidth(イメージキットコントロール／FileIO プロパティ)

【機能】
ベクトルイメージ読み込み時の高さと幅を取得または設定します。

【書式】
※**VectorHeight** にて説明(**VectorWidth** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**VectorHeight** [= *int*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**VectorHeight** [= *Integer*]

【設定値】
VectorHeight はイメージの高さ(ピクセル)、デフォルトは 0。
VectorWidth はイメージの幅(ピクセル)、デフォルトは 0。

【解説】
DXF,SXFの場合
VectorHeight もしくは **VectorWidth** が 0 以下の場合は高さ 600 幅 800 の値を使用します。
EMF,SVG,WMFの場合
VectorHeight もしくは **VectorWidth** が 0 以下の場合はファイルに格納されているサイズを使用します。

(注意)
読み込まれるイメージの縦横のサイズは、設定した矩形内(VectorWidth * VectorHeight)で縦横比を考慮した最も大きなサイズとなりますので、設定したサイズ通りにならない場合があります。

【値の設定】 実行時

【値の参照】 実行時

CMYKBmpPlaneFileLoad（イメージキットコントロール／FileIO メソッド）

【機能】

CMYK プレーン毎に保存してある BMP ファイルからラスタイメージを読み込みます。

【書式】

(1)C++Builder

[*bool =*]*imagekitcontrolname*->**FileIO->CMYKBmpPlaneFileLoad**(const UnicodeString CFileName, const UnicodeString MFileName, const UnicodeString YFileName, const UnicodeString KFileName)

(2)Delphi

[*Boolean =*]*imagekitcontrolname*.**FileIO.CMYKBmpPlaneFileLoad**(const CFileName: string; const MFileName: string; const YFileName: string; const KFileName: string)

【引数】

名称	内容
CFileName	C プレーンとして読み込むファイル名
MFileName	M プレーンとして読み込むファイル名
YFileName	Y プレーンとして読み込むファイル名
KFileName	K プレーンとして読み込むファイル名

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

CMYK それぞれのプレーンを表すイメージは 8 ビットグレーで、出力イメージは 24 ビットカラーです。

成功すると **LayerNo** プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）にイメージのメモリハンドルが設定されます。

CMYKBmpPlaneFileSave（イメージキットコントロール／FileIO メソッド）

【機能】

ラスタイメージを CMYK プレーン毎に BMP 形式でファイルに保存します。

【書式】

(1)C++Builder

[*bool =*]*imagekitcontrolname*->**FileIO->CMYKBmpPlaneFileSave**(const UnicodeString CFileName, const UnicodeString MFileName, const UnicodeString YFileName, const UnicodeString KFileName, NativeUInt ImageHandle)

[*bool =*]*imagekitcontrolname*->**FileIO->CMYKBmpPlaneFileSave**(const UnicodeString CFileName, const UnicodeString MFileName, const UnicodeString YFileName, const UnicodeString KFileName)

(2)Delphi

[*Boolean =*]*imagekitcontrolname*.**FileIO.CMYKBmpPlaneFileSave**(const CFileName: string; const MFileName: string; const YFileName: string; const KFileName: string; ImageHandle: THandle)

[*Boolean =*]*imagekitcontrolname*.**FileIO.CMYKBmpPlaneFileSave**(const CFileName: string; const MFileName: string; const YFileName: string; const KFileName: string)

【引数】

名称	内容
CFileName	C プレーンとして保存するファイル名
MFileName	M プレーンとして保存するファイル名
YFileName	Y プレーンとして保存するファイル名
KFileName	K プレーンとして保存するファイル名
ImageHandle	保存対象となるイメージのメモリハンドル

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

引数のImageHandleに有効なメモリハンドルを与えた場合

ImageHandle に設定されたメモリハンドルを CMYK のそれぞれのプレーン毎にファイルに保存します。

ImageHandleが不要なメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合

LayerNo プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されたメモリハンドルを CMYK のそれぞれのプレーン毎にファイルに保存します。

※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

保存対象イメージは 24 ビットカラーで、保存されるイメージは 8 ビットグレーとなります。

FileLoadAsRawData（イメージキットコントロール／FileIO メソッド）

【機能】
ファイルからそのままの状態でイメージを読み込みます。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**FileLoadAsRawData**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FileLoadAsRawData**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
FileName プロパティに設定されたファイルを読み込み、**ImageHandleRawData** プロパティに Raw データを設定します。
読み込み処理中の進捗状況ダイアログを表示させるためにキャプション、メッセージ、ボタンをそれぞれ
Caption,**Message**,**ButtonName** プロパティに設定することもできます。

FileSaveAsRawData(イメージキットコントロール／FileIO メソッド)

【機能】
Raw データをファイルに保存します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**FileSaveAsRawData**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FileSaveAsRawData**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ImageHandleRawData プロパティに設定された Raw データを **FileName** プロパティに設定されたファイルに保存します。
保存処理中の進捗状況ダイアログを表示させるためにキャプション、メッセージ、ボタンをそれぞれ
Caption,**Message**,**ButtonName** プロパティに設定することもできます。

FTPConnect（イメージキットコントロール／FileIO メソッド）

【機能】
FTP サーバに接続します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPConnect**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, bool PassiveMode, const UnicodeString UserName, const UnicodeString Password)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPConnect**(const ServerName: string; const ProxyName: string; PortNo: Smallint; PassiveMode: Boolean; const UserName: string; const Password: string)

【引数】

名称	内容
ServerName	FTP サーバの名称もしくは IP アドレス
ProxyName	プロキシサーバの IP アドレス
PortNo	ポート番号（デフォルトは 21）
PassiveMode	パッシブモード接続の有無 [False:アクティブモードで接続, True:パッシブモードで接続]
UserName	ユーザ名
Password	パスワード

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。
当メソッドで FTP サーバへ接続し、**FTPDeleteFileEx**,**FTPGetFileEx**,**FTPPutFileEx**,**FTPRenameFileEx** メソッドを使用します。
プロキシサーバを使用しない場合は、ProxyName に空文字列を渡してください。

FTPDeleteFile（イメージキットコントロール／FileIO メソッド）

【機能】
FTP サーバに存在するファイルを削除します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPDeleteFile**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, bool PassiveMode, const UnicodeString RemoteFilePath, const UnicodeString UserName, const UnicodeString Password)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPDeleteFile**(const ServerName: string; const ProxyName: string; PortNo: Smallint; PassiveMode: Boolean; const RemoteFilePath: string; const UserName: string; const Password: string)

【引数】

名称	内容
ServerName	FTP サーバの名称もしくは IP アドレス
ProxyName	プロキシサーバの IP アドレス
PortNo	ポート番号（デフォルトは 21）
PassiveMode	パッシブモード接続の有無 ［ False:アクティブモードで接続, True:パッシブモードで接続 ］
RemoteFilePath	削除するファイル名
UserName	ユーザ名
Password	パスワード

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。当メソッドでは FTP サーバへの接続と切断を行います。
プロキシサーバを使用しない場合は、ProxyName に空文字列を渡してください。

FTPDeleteFileEx(イメージキットコントロール／FileIO メソッド)

【機能】
FTP サーバに存在するファイルを削除します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPDeleteFileEx**(const UnicodeString RemoteFilePath)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPDeleteFileEx**(const RemoteFilePath: string)

【引数】

名称	内容
RemoteFilePath	削除するファイル名

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。**FTPConnect** メソッドで FTP サーバに接続してから使用します。

コード例:
(1)C++Builder
```
if (VImageKit1->FileIO->FTPConnect("www.newtone.co.jp", "", 21, true, "User", "Password") == false) return;
VImageKit1->FileIO->FTPDeleteFileEx("images/001.jpg");
VImageKit1->FileIO->FTPDisconnect();
```
(2)Delphi
```
if (VImageKit1.FileIO.FTPConnect('www.newtone.co.jp', '', 21, True, 'User', 'Password') = False) then Exit;
VImageKit1.FileIO.FTPDeleteFileEx('images/001.jpg');
VImageKit1.FileIO.FTPDisconnect();
```

FTPDisconnect（イメージキットコントロール／FileIO メソッド）

【機能】
接続している FTP サーバを切断します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPDisconnect**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPDisconnect**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。**FTPConnect** メソッドで接続した FTP サーバを切断します。

FTPGetFile（イメージキットコントロール／FileIO メソッド）

【機能】
FTP サーバからファイルを取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPGetFile**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, bool PassiveMode, const UnicodeString RemoteFilePath, const UnicodeString UserName, const UnicodeString Password, int TransPercent)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPGetFile**(const ServerName: string; const ProxyName: string; PortNo: Smallint; PassiveMode: Boolean; const RemoteFilePath: string; const UserName: string; const Password: string; TransPercent: Integer)

【引数】

名称	内容
ServerName	FTP サーバの名称もしくは IP アドレス
ProxyName	プロキシサーバの IP アドレス
PortNo	ポート番号（デフォルトは 21）
PassiveMode	パッシブモード接続の有無 [False:アクティブモードで接続, True:パッシブモードで接続]
RemoteFilePath	FTP サーバから取得するファイル名
UserName	ユーザ名
Password	パスワード
TransPercent	転送単位％（1～100）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。当メソッドでは FTP サーバへの接続と切断を行います。TransPercent には転送単位を％で与えます。
取得したファイルは **FileName** プロパティに設定されたファイルに保存されます。
プロキシサーバを使用しない場合は、ProxyName に空文字列を渡してください。

FTPGetFileEx(イメージキットコントロール／FileIO メソッド)

【機能】
FTP サーバからファイルを取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPGetFileEx**(cconst UnicodeString RemoteFilePath, int TransPercent)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPGetFileEx**(const RemoteFilePath: string; TransPercent: Integer)

【引数】

名称	内容
RemoteFilePath	FTP サーバから取得するファイル名
TransPercent	転送単位％(1～100)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。**FTPConnect** メソッドで FTP サーバに接続してから使用します。TransPercent には転送単位を％で与えます。
取得したファイルは **FileName** プロパティに設定されたファイルに保存されます。

コード例:
(1)C++Builder
```
if (VImageKit1->FileIO->FTPConnect("www.newtone.co.jp", "", 21, true, "User", "Password") == false) return;
VImageKit1->FileIO->FileName = "C:¥¥Images¥¥001.jpg";
VImageKit1->FileIO->FTPGetFileEx("images/001.jpg", 10);
VImageKit1->FileIO->FTPDisconnect();
```
(2)Delphi
```
if (VImageKit1.FileIO.FTPConnect('www.newtone.co.jp', '', 21, True, 'User', 'Password') = False) then Exit;
VImageKit1.FileIO.FileName := 'C:¥Images¥001.jpg';
VImageKit1.FileIO.FTPGetFileEx('images/001.jpg', 10);
VImageKit1.FileIO.FTPDisconnect();
```

FTPPutFile（イメージキットコントロール／FileIO メソッド）

【機能】
FTP サーバへファイルを転送します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPPutFile**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, bool PassiveMode, const UnicodeString RemoteFilePath, const UnicodeString UserName, const UnicodeString Password, int TransPercent)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPPutFile**(const ServerName: string; const ProxyName: string; PortNo: Smallint; PassiveMode: Boolean; const RemoteFilePath: string; const UserName: string; const Password: string; TransPercent: Integer)

【引数】

名称	内容
ServerName	FTP サーバの名称もしくは IP アドレス
ProxyName	プロキシサーバの IP アドレス
PortNo	ポート番号（デフォルトは 21）
PassiveMode	パッシブモード接続の有無 [False:アクティブモードで接続, True:パッシブモードで接続]
RemoteFilePath	転送先（FTP サーバ）ファイル名
UserName	ユーザ名
Password	パスワード
TransPercent	転送単位％（1～100）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。当メソッドでは FTP サーバへの接続と切断を行います。TransPercent には転送単位を％で与えます。
転送ファイルは **FileName** プロパティに設定されたファイルとなります。
プロキシサーバを使用しない場合は、ProxyName に空文字列を渡してください。

FTPPutFileEx(イメージキットコントロール／FileIO メソッド)

【機能】
FTP サーバへファイルを転送します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPPutFileEx**(const UnicodeString RemoteFilePath, int TransPercent)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPPutFileEx**(const RemoteFilePath: string; TransPercent: Integer)

【引数】

名称	内容
RemoteFilePath	転送先(FTP サーバ)ファイル名
TransPercent	転送単位%(1～100)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。**FTPConnect** メソッドで FTP サーバに接続してから使用します。TransPercent には転送単位を%で与えます。
転送ファイルは **FileName** プロパティに設定されたファイルとなります。

コード例:
(1)C++Builder
```
if (VImageKit1->FileIO->FTPConnect("www.newtone.co.jp", "", 21, true, "User", "Password") == false) return;
VImageKit1->FileIO->FileName = "C:¥¥Images¥¥001.jpg";
VImageKit1->FileIO->FTPPutFileEx("images/001.jpg", 10);
VImageKit1->FileIO->FTPDisconnect();
```
(2)Delphi
```
if (VImageKit1.FileIO.FTPConnect('www.newtone.co.jp', '', 21, True, 'User', 'Password') = False) then Exit;
VImageKit1.FileIO.FileName := 'C:¥Images¥001.jpg';
VImageKit1.FileIO.FTPPutFileEx('images/001.jpg', 10);
VImageKit1.FileIO.FTPDisconnect();
```

FTPRenameFile（イメージキットコントロール／FileIO メソッド）

【機能】
FTP サーバに存在するファイルの名称を変更します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPRenameFile**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, bool PassiveMode, const UnicodeString RemoteOldFilePath, const UnicodeString RemoteNewFile, const UnicodeString UserName, const UnicodeString Password)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPRenameFile**(const ServerName: string; const ProxyName: string; PortNo: Smallint; PassiveMode: Boolean; const RemoteOldFilePath: string; const RemoteNewFile: string; const UserName: string; const Password: string)

【引数】

名称	内容
ServerName	FTP サーバの名称もしくは IP アドレス
ProxyName	プロキシサーバの IP アドレス
PortNo	ポート番号（デフォルトは 21）
PassiveMode	パッシブモード接続の有無 [False:アクティブモードで接続, True:パッシブモードで接続]
RemoteOldFilePath	変更前のファイル名
RemoteNewFile	変更後のファイル名
UserName	ユーザ名
Password	パスワード

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。当メソッドでは FTP サーバへの接続と切断を行います。
プロキシサーバを使用しない場合は、ProxyName に空文字列を渡してください。

FTPRenameFileEx（イメージキットコントロール／FileIO メソッド）

【機能】
FTP サーバに存在するファイルの名称を変更します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**FTPRenameFileEx**(const UnicodeString RemoteOldFilePath, const UnicodeString RemoteNewFile)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**FTPRenameFileEx**(const RemoteOldFilePath: string; const RemoteNewFile: string)

【引数】

名称	内容
RemoteOldFilePath	変更前のファイル名
RemoteNewFile	変更後のファイル名

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。**FTPConnect** メソッドで FTP サーバに接続してから使用します。

コード例：
(1)C++Builder
```
if (VImageKit1->FileIO->FTPConnect("www.newtone.co.jp", "", 21, true, "User", "Password") == false) return;
VImageKit1->FileIO->FTPRenameFileEx("images/001.jpg", "002.jpg");
VImageKit1->FileIO->FTPDisconnect();
```
(2)Delphi
```
if (VImageKit1.FileIO.FTPConnect('www.newtone.co.jp', '', 21, True, 'User', 'Password') = False) then Exit;
VImageKit1.FileIO.FTPRenameFileEx('images/001.jpg', '002.jpg');
VImageKit1.FileIO.FTPDisconnect();
```

GetImageFileType（イメージキットコントロール／FileIO メソッド）

【機能】

ファイルからイメージの各種情報を次のプロパティに設定します。

Comment,CreationTimeDay,CreationTimeHour,CreationTimeMinute,CreationTimeMonth,CreationTimeSecond,CreationTimeYear,Exif,FileBitCount,FileHeight,FileImageSize,FileMaxPage,FileSize,FileType,FileWidth,FileWidthByte,FileXdpi,FileYdpi,Interlace,LastAccessTimeDay,LastAccessTimeHour,LastAccessTimeMinute,LastAccessTimeMonth,LastAccessTimeSecond,LastAccessTimeYear,LastWriteTimeDay,LastWriteTimeHour,LastWriteTimeMinute,LastWriteTimeMonth,LastWriteTimeSecond,LastWriteTimeYear。

【書式】

(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO->GetImageFileType**()

(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO.GetImageFileType**

【引数】

ありません。

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

このメソッドを実行するには、予め **FileName** プロパティと **LoadPage** プロパティに適切な値が設定されている必要があります。**FileMaxPage,FileSize** 以外のプロパティは、**LoadPage** プロパティで設定されたページの情報となります。ただし、TIFF 形式を除く **Comment** プロパティはファイル単位の情報となります。

対応しているファイル形式であっても、ファイルに該当する情報が保存されていない場合はプロパティに 0 もしくは空文字列が設定されます。設定されるプロパティについては、各プロパティの説明を参照してください。

GetImageFileTypeMem（イメージキットコントロール／FileIO メソッド）

【機能】

Raw データからイメージの各種情報を次のプロパティに設定します。
Comment,**Exif**,**FileBitCount**,**FileHeight**,**FileImageSize**,**FileMaxPage**,**FileSize**,**FileType**,**FileWidth**,**FileWidthByte**,**FileXdpi**,**FileYdpi**,**Interlace**。

【書式】

(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**GetImageFileTypeMem**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**GetImageFileTypeMem**

【引数】

ありません。

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

このメソッドを実行するには、予め **ImageHandleRawData** プロパティと **LoadPage** プロパティに適切な値が設定されている必要があります。
FileMaxPage,**FileSize** 以外のプロパティは、**LoadPage** プロパティで設定されたページの情報となります。ただし、TIFF 形式を除く **Comment** プロパティはファイル単位の情報となります。
対応しているファイル形式であっても、ファイルに該当する情報が保存されていない場合はプロパティに 0 もしくは空文字列が設定されます。設定されるプロパティについては、各プロパティの説明を参照してください。
ファイルあるいは Raw データから情報を取得する違いはありますが、動作としては **GetImageFileType** メソッドと同じです。ただし、ファイルの作成日時・アクセス日時・更新日時を示すプロパティには 0 が設定されます。

HTTPConnect（イメージキットコントロール／FileIO メソッド）

【機能】
HTTP(S)サーバに接続します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**HTTPConnect**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, bool HTTPS, const UnicodeString UserName, const UnicodeString Password)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**HTTPConnect**(const ServerName: string; const ProxyName: string; PortNo: Smallint; HTTPS: Boolean; const UserName: string; const Password: string)

【引数】

名称	内容
ServerName	HTTP(S)サーバの名称もしくは IP アドレス
ProxyName	プロキシサーバの IP アドレス
PortNo	ポート番号（デフォルトは HTTP が 80、HTTPS が 443）
HTTPS	接続先サーバの種類 [False:HTTP, True:HTTPS]
UserName	ユーザ名
Password	パスワード

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。当メソッドで HTTP(S)サーバへ接続し、**HTTPGetFileEx**,**HTTPPutFileEx** メソッドを使用します。プロキシサーバを使用しない場合は、ProxyName に空文字列を渡してください。

HTTPDisconnect（イメージキットコントロール／FileIO メソッド）

【機能】
接続している HTTP(S)サーバを切断します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**HTTPDisconnect**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**HTTPDisconnect**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。**HTTPConnect** メソッドで接続した HTTP(S)サーバを切断します。

HTTPGetFile,HTTPSGetFile（イメージキットコントロール／FileIO メソッド）

【機能】
HTTP(S)サーバからファイルを取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**HTTPGetFile**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, const UnicodeString RemoteFilePath, const UnicodeString UserName, const UnicodeString Password, int TransPercent)
[*bool =*]*imagekitcontrolname*->**FileIO**->**HTTPSGetFile**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, const UnicodeString RemoteFilePath, const UnicodeString UserName, const UnicodeString Password, int TransPercent)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**HTTPGetFile**(const ServerName: string; const ProxyName: string; PortNo: Smallint; const RemoteFilePath: string; const UserName: string; const Password: string; TransPercent: Integer)
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**HTTPSGetFile**(const ServerName: string; const ProxyName: string; PortNo: Smallint; const RemoteFilePath: string; const UserName: string; const Password: string; TransPercent: Integer)

【引数】

名称	内容
ServerName	HTTP(S)サーバの名称もしくは IP アドレス
ProxyName	プロキシサーバの IP アドレス
PortNo	ポート番号（デフォルトは HTTP が 80、HTTPS が 443）
RemoteFilePath	HTTP(S)サーバから取得するファイル名
UserName	ユーザ名
Password	パスワード
TransPercent	転送単位％（1～100）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。当メソッドでは HTTP(S)サーバへの接続と切断を行います。TransPercent には転送単位を％で与えます。
取得したファイルは **FileName** プロパティに設定されたファイルに保存されます。
プロキシサーバを使用しない場合は、ProxyName に空文字列を渡してください。
HTTP サーバを利用する場合は **HTTPGetFile** メソッドを、HTTPS サーバを利用する場合は **HTTPSGetFile** メソッドを使用してください。

HTTPGetFileEx（イメージキットコントロール／FileIO メソッド）

【機能】
HTTP(S)サーバからファイルを取得します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**FileIO**->**HTTPGetFileEx**(const UnicodeString RemoteFilePath, int TransPercent)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**FileIO**.**HTTPGetFileEx**(const RemoteFilePath: string; TransPercent: Integer)

【引数】

名称	内容
RemoteFilePath	HTTP(S)サーバから取得するファイル名
TransPercent	転送単位％（1～100）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。**HTTPConnect** メソッドで HTTP(S)サーバに接続してから使用します。TransPercent には転送単位を％で与えます。
取得したファイルは **FileName** プロパティに設定されたファイルに保存されます。

コード例：
(1)C++Builder
```
if (VImageKit1->FileIO->HTTPConnect("www.newtone.co.jp", "", 443, true, "User", "Password") == false) return;
VImageKit1->FileIO->FileName = "C:¥¥Images¥¥001.jpg";
VImageKit1->FileIO->HTTPGetFileEx("images/001.jpg", 10);
VImageKit1->FileIO->HTTPDisconnect();
```
(2)Delphi
```
if (VImageKit1.FileIO.HTTPConnect('www.newtone.co.jp', '', 443, True, 'User', 'Password') = False) then Exit;
VImageKit1.FileIO.FileName := 'C:¥Images¥001.jpg';
VImageKit1.FileIO.HTTPGetFileEx('images/001.jpg', 10);
VImageKit1.FileIO.HTTPDisconnect();
```

HTTPPutFile,HTTPSPutFile（イメージキットコントロール／FileIO メソッド）

【機能】
HTTP(S)サーバへファイルを転送します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**HTTPPutFile**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, const UnicodeString RemoteFilePath, const UnicodeString UserName, const UnicodeString Password, int TransPercent)
[*bool =*]*imagekitcontrolname*->**FileIO**->**HTTPSPutFile**(const UnicodeString ServerName, const UnicodeString ProxyName, short PortNo, const UnicodeString RemoteFilePath, const UnicodeString UserName, const UnicodeString Password, int TransPercent)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**HTTPPutFile**(const ServerName: string; const ProxyName: string; PortNo: Smallint; const RemoteFilePath: string; const UserName: string; const Password: string; TransPercent: Integer)
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**HTTPSPutFile**(const ServerName: string; const ProxyName: string; PortNo: Smallint; const RemoteFilePath: string; const UserName: string; const Password: string; TransPercent: Integer)

【引数】

名称	内容
ServerName	HTTP(S)サーバの名称もしくは IP アドレス
ProxyName	プロキシサーバの IP アドレス
PortNo	ポート番号（デフォルトは HTTP が 80、HTTPS が 443）
RemoteFilePath	転送先（HTTP(S)サーバ）ファイル名
UserName	ユーザ名
Password	パスワード
TransPercent	転送単位％（1～100）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。当メソッドでは HTTP(S)サーバへの接続と切断を行います。TransPercent には転送単位を％で与えます。
転送ファイルは **FileName** プロパティに設定されたファイルとなります。
プロキシサーバを使用しない場合は、ProxyName に空文字列を渡してください。
HTTP サーバを利用する場合は **HTTPPutFile** メソッドを、HTTPS サーバを利用する場合は **HTTPSPutFile** メソッドを使用してください。

HTTPPutFileEx（イメージキットコントロール／FileIO メソッド）

【機能】
HTTP(S)サーバへファイルを転送します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**HTTPPutFileEx**(const UnicodeString RemoteFilePath, int TransPercent)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**HTTPPutFileEx**(const RemoteFilePath: string; TransPercent: Integer)

【引数】

名称	内容
RemoteFilePath	転送先（HTTP(S)サーバ）ファイル名
TransPercent	転送単位％（1～100）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
クライアントからの使用に限ります。**HTTPConnect** メソッドで HTTP(S)サーバに接続してから使用します。TransPercent には転送単位を％で与えます。
転送ファイルは **FileName** プロパティに設定されたファイルとなります。

コード例：
(1)C++Builder
```
if (VImageKit1->FileIO->HTTPConnect("www.newtone.co.jp", "", 443, true, "User", "Password") == false) return;
VImageKit1->FileIO->FileName = "C:¥¥Images¥¥001.jpg";
VImageKit1->FileIO->HTTPPutFileEx("images/001.jpg", 10);
VImageKit1->FileIO->HTTPDisconnect();
```
(2)Delphi
```
if (VImageKit1.FileIO.HTTPConnect('www.newtone.co.jp', '', 443, True, 'User', 'Password') = False) then Exit;
VImageKit1.FileIO.FileName := 'C:¥Images¥001.jpg';
VImageKit1.FileIO.HTTPPutFileEx('images/001.jpg', 10);
VImageKit1.FileIO.HTTPDisconnect();
```

LoadFile(イメージキットコントロール／FileIO メソッド)

【機能】
ファイルからイメージを読み込みます。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**LoadFile**(TVIkLoadFile LoadType)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**LoadFile**(LoadType: TVIkLoadFile)

【TVIkLoadFile 型】
ユニット
IkInit
type
TVIkLoadFile = (vikLoad, vikLoadBMP, vikLoadJPEG, vikLoadGIF, vikLoadTIFF, vikLoadPNG, vikLoadFPX, vikLoadPCX, vikLoadWMF, vikLoadEMF, vikLoadDXF, vikLoadSVG, vikLoadJPEG2000, vikLoadSXFP21, vikLoadSXFSFC);

【引数】

名称	内容
LoadType	vikLoad(自動認識) vikLoadBMP(BMP) vikLoadJPEG(JPEG/Exif) vikLoadGIF(GIF) vikLoadTIFF(TIFF) vikLoadPNG(PNG) vikLoadFPX(FPX) vikLoadPCX(PCX) vikLoadWMF(WMF) vikLoadEMF(EMF) vikLoadDXF(DXF) vikLoadSVG(SVG) vikLoadJPEG2000(JPEG2000) vikLoadSXFP21(SXFP21) vikLoadSXFSFC(SXFSFC)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
FileName プロパティに設定されたファイルを読み込み、**LayerNo** プロパティが示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)にイメージのメモリハンドルが設定されます。**ただし、TIFF の JPEG 形式については読み込みに失敗するファイルも存在しますのでご了承ください。**
マルチイメージファイルの場合は、**LoadPage** プロパティに読み込むページを設定します。
PNG ファイルの場合は **PngAlphaChannel** プロパティを設定してください。
DXF,EMF,SVG,SXF,WMF ファイルの場合は **VectorHeight**,**VectorWidth** プロパティを設定してください。
また、読み込み処理中の進捗状況ダイアログを表示させるためにキャプション、メッセージ、ボタンをそれぞれ **Caption**,**Message**,**ButtonName** プロパティに設定することもできます。

Exif(JPEG)の主画像を読み込む場合は当メソッドを使用しますが、サムネイル画像や詳細な情報を取得する場合は、**GetImageFileType** メソッドをご使用ください。Exif はバージョン 2.3 の一部のタグにも対応しています。

＜FlashPix について＞
FlashPix の最大の特徴は、マルチ・レゾリューション(複数解像度)で、さらにタイル構造を持つファイル・フォーマットであり、基画像から最小画像までの、画素数の異なる複数画面を一つのファイルの中に持つことができ、編集・印刷など異なった用途に最適な解像度で対応することができます。それぞれの解像度の画面は、64x64 のブロック(タイル)に分割されています。タイル内は、非圧縮・JPEG 圧縮・単色圧縮のいずれかが可能です。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikLoad, ikLoadBMP, ikLoadJPEG, ikLoadGIF, ikLoadTIFF, ikLoadPNG, ikLoadFPX, ikLoadPCX, ikLoadWMF, ikLoadEMF, ikLoadDXF, ikLoadSVG, ikLoadJPEG2000, ikLoadSXFP21, ikLoadSXFSFC)。

【ImageKit7 ActiveX/VCL との違い】
FTP サーバーや HTTP(S)サーバーに配置されているファイルを読み込むことができるようになりました。詳しくは **FileName** プロパティの説明を参照してください。

LoadFileMem（イメージキットコントロール／FileIO メソッド）

【機能】
Raw データからイメージを読み込みます。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**LoadFileMem**(TVIkLoadFile LoadType)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**LoadFileMem**(LoadType: TVIkLoadFile)

【TVIkLoadFile 型】
ユニット
IkInit
type
TVIkLoadFile = (vikLoad, vikLoadBMP, vikLoadJPEG, vikLoadGIF, vikLoadTIFF, vikLoadPNG, vikLoadFPX, vikLoadPCX, vikLoadWMF, vikLoadEMF, vikLoadDXF, vikLoadSVG, vikLoadJPEG2000, vikLoadSXFP21, vikLoadSXFSFC);

【引数】

名称	内容
LoadType	vikLoad(自動認識) vikLoadBMP(BMP) vikLoadJPEG(JPEG/Exif) vikLoadGIF(GIF) vikLoadTIFF(TIFF) vikLoadPNG(PNG) vikLoadFPX(FPX) vikLoadPCX(PCX) vikLoadWMF(WMF) vikLoadEMF(EMF) vikLoadDXF(DXF) vikLoadSVG(SVG) vikLoadJPEG2000(JPEG2000) vikLoadSXFP21(SXFP21) vikLoadSXFSFC(SXFSFC)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ImageHandleRawData プロパティに設定された Raw データを、**LayerNo** プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）にイメージのメモリハンドルが設定されます。**ただし、TIFF の JPEG 形式については読み込みに失敗する場合がありますのでご了承ください。**
Raw データがマルチイメージの場合は、**LoadPage** プロパティに読み込むページを設定します。
PNG 形式の Raw データの場合は **PngAlphaChannel** プロパティを設定してください。
DXF,EMF,SVG,SXF,WMF 形式の Raw データの場合は **VectorHeight**,**VectorWidth** プロパティを設定してください。
また、読み込み処理中の進捗状況ダイアログを表示させるためにキャプション、メッセージ、ボタンをそれぞれ **Caption**,**Message**,**ButtonName** プロパティに設定することもできます。
ファイルあるいは Raw データから読み込む違いはありますが、動作としては **LoadFile** メソッドと同じです。
（注意）
FPX,SXF 形式の場合、内部で一時的にテンポラリファイルを作成します。

Exif(JPEG)の主画像を読み込む場合は当メソッドを使用しますが、サムネイル画像や詳細な情報を取得する場合は、**GetImageFileTypeMem** メソッドをご使用ください。Exif はバージョン 2.3 の一部のタグにも対応しています。

【ImageKit7/8/9/10 ActiveX との違い】

列挙型の識別子の先頭に v が付加されました(ActiveX は ikLoad, ikLoadBMP, ikLoadJPEG, ikLoadGIF, ikLoadTIFF, ikLoadPNG, ikLoadFPX, ikLoadPCX, ikLoadWMF, ikLoadEMF, ikLoadDXF, ikLoadSVG, ikLoadJPEG2000, ikLoadSXFP21, ikLoadSXFSFC)。

LoadFromStream（イメージキットコントロール／FileIO メソッド）

【機能】
ストリームからイメージを読み込みます。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**LoadFromStream**(TStream* Stream)
(2)Delphi　*imagekitcontrolname*.**FileIO**.**LoadFromStream**(Stream: TStream)

【引数】

名称	内容
Stream	ストリーム

【戻り値】
ありません。

【解説】
メソッドを実行すると **ImageHandleRawData** プロパティに Raw データが設定されます。

コード例：
(1)C++Builder

```
TMemoryStream *Stream;

Stream = new TMemoryStream();

// File -> Rawdata
VImageKit1->FileIO->FileName = "c:¥¥001.jpg"
VImageKit1->FileIO->FileLoadAsRawData();

// Rawdata -> Stream
VImageKit1->FileIO->SaveToStream(Stream);

// Clear Rawdata
VImageKit1->FileIO->ImageHandleRawData = 0;

// Stream -> Rawdata
Stream->Position = 0; //  ストリームの先頭にリセット
VImageKit1->FileIO->LoadFromStream(Stream);

// Rawdata -> File
VImageKit1->FileIO->FileName = "c:¥¥abc.jpg"
VImageKit1->FileIO->FileSaveAsRawData();

delete Stream;
```

(2)Delphi

```
Stream: TMemoryStream;

Stream := TMemoryStream.Create;

// File -> Rawdata
VImageKit1.FileIO.FileName := 'c:¥001.jpg';
VImageKit1.FileIO.FileLoadAsRawData;

// Rawdata -> Stream
VImageKit1.FileIO.SaveToStream(Stream);
```

```
// Clear Rawdata
VImageKit1.FileIO.ImageHandleRawData := 0;

// Stream -> Rawdata
Stream.Position := 0; //  ストリームの先頭にリセット
VImageKit1.FileIO.LoadFromStream(Stream);

// Rawdata -> File
VImageKit1.FileIO.FileName := 'c:¥abc.jpg';
VImageKit1.FileIO.FileSaveAsRawData;

Stream.Free;
```

【ImageKit7/8/9/10 ActiveX との違い】
引数の型が変更されました。

OpenFileDialog(イメージキットコントロール／FileIO メソッド)

【機能】
プレビュー付きのファイルオープンダイアログを表示します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**OpenFileDialog**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**OpenFileDialog**

【引数】
ありません。

【戻り値】
開くを選択した場合は True、キャンセルの場合は False を返します。

【解説】
このメソッドを実行するには、予め **FilePath** プロパティと **FileExt** プロパティに適切な値が設定されている必要があります。
メソッド実行前に **ExtendedDialog** プロパティに True を設定すると、ダイアログ初期表示時にプレビューとファイル情報表示のチェックボックスが表示されます。そのため、**Preview** と **Infomation** プロパティが False でもダイアログ表示時にそれぞれ表示することができます。
開くボタンを選択するとファイル名がフルパスで **FileName** プロパティに設定されます。

RGBBmpPlaneFileLoad（イメージキットコントロール／FileIO メソッド）

【機能】

RGB プレーン毎に保存してある BMP ファイルからラスタイメージを読み込みます。

【書式】

(1)C++Builder

[*bool =*]*imagekitcontrolname*->**FileIO**->**RGBBmpPlaneFileLoad**(const UnicodeString RedFileName, const UnicodeString GreenFileName, const UnicodeString BlueFileName)

(2)Delphi

[*Boolean =*]*imagekitcontrolname*.**FileIO**.**RGBBmpPlaneFileLoad**(const RedFileName: string; const GreenFileName: string; const BlueFileName: string)

【引数】

名称	内容
RedFileName	R プレーンとして読み込むファイル名
GreenFileName	G プレーンとして読み込むファイル名
BlueFileName	B プレーンとして読み込むファイル名

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

RGB それぞれのプレーンを表すイメージは 8 ビットグレーで、出力イメージは 24 ビットカラーです。成功すると **LayerNo** プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）にイメージのメモリハンドルが設定されます。

RGBBmpPlaneFileSave（イメージキットコントロール／FileIO メソッド）

【機能】
ラスタイメージを RGB プレーン毎に BMP 形式でファイルに保存します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**RGBBmpPlaneFileSave**(const UnicodeString RedFileName, const UnicodeString GreenFileName, const UnicodeString BlueFileName, NativeUInt ImageHandle)
[*bool =*]*imagekitcontrolname*->**FileIO**->**RGBBmpPlaneFileSave**(const UnicodeString RedFileName, const UnicodeString GreenFileName, const UnicodeString BlueFileName)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**RGBBmpPlaneFileSave**(const RedFileName: string; const GreenFileName: string; const BlueFileName: string; ImageHandle: THandle)
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**RGBBmpPlaneFileSave**(const RedFileName: string; const GreenFileName: string; const BlueFileName: string)

【引数】

名称	内容
RedFileName	R プレーンとして保存するファイル名
GreenFileName	G プレーンとして保存するファイル名
BlueFileName	B プレーンとして保存するファイル名
ImageHandle	保存対象となるイメージのメモリハンドル

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルを RGB のそれぞれのプレーン毎にファイルに保存します。
ImageHandleが不要なメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されたメモリハンドルを RGB のそれぞれのプレーン毎にファイルに保存します。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

保存対象イメージは 24 ビットカラーで、保存されるイメージは 8 ビットグレーとなります。

SaveFile(イメージキットコントロール／FileIO メソッド)

【機能】
イメージをファイルへ保存します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**SaveFile**(TVIkSaveFile SaveType, NativeUInt ImageHandle)
[*bool =*]*imagekitcontrolname*->**FileIO**->**SaveFile**(TVIkSaveFile SaveType)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**SaveFile**(SaveType: TVIkSaveFile; ImageHandle: THandle)
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**SaveFile**(SaveType: TVIkSaveFile)

【TVIkSaveFile 型】
ユニット
IkInit
type
TVIkSaveFile = (vikSaveBMP, vikSaveBMPRLE, vikSaveJPEG, vikSaveJPEGProgress, vikSaveGIF, vikSaveTIFFNoncompressed, vikSaveTIFFCcittrle, vikSaveTIFFGroup3_1D, vikSaveTIFFGroup3_2D, vikSaveTIFFGroup4, vikSaveTIFFPackbits, vikSaveTIFFLZW, vikSavePNG, vikSaveFPXNoncompressed, vikSaveFPXSingleColor, vikSaveFPXJpeg, vikSavePCX, vikSaveWMF, vikSaveEMF, vikSaveDXF, vikSaveSVG, vikSaveJPEG2000, vikSaveJPEG2000Stream, vikSaveSXFP21, vikSaveSXFSFC, vikSaveTIFFJPEG, vikSaveJPEGExif, vikSaveJPEGProgressExif);

【引数】

名称	内容
SaveType	vikSaveBMP(BMP 非圧縮) vikSaveBMPRLE(BMP 圧縮) vikSaveJPEG(JPEG 基本 DCT) vikSaveJPEGProgress(JPEG プログレッシブ DCT) vikSaveGIF(GIF) vikSaveTIFFNoncompressed(TIFF 非圧縮) vikSaveTIFFCcittrle(TIFF CCITTRLE) vikSaveTIFFGroup3_1D(TIFF GROUP3-1D) vikSaveTIFFGroup3_2D(TIFF GROUP3-2D) vikSaveTIFFGroup4(TIFF GROUP4) vikSaveTIFFPackbits(TIFF PACKBITS) vikSaveTIFFLZW(TIFF LZW) vikSavePNG(PNG) vikSaveFPXNoncompressed(FPX 非圧縮) vikSaveFPXSingleColor(FPX 単色圧縮) vikSaveFPXJpeg(FPX JPEG 圧縮) vikSavePCX(PCX) vikSaveWMF(WMF) vikSaveEMF(EMF) vikSaveDXF(DXF) vikSaveSVG(SVG) vikSaveJPEG2000(JPEG2000 Part1) vikSaveJPEG2000Stream(JPEG2000 Code Stream) vikSaveSXFP21(SXF p21) vikSaveSXFSFC(SXF sfc) vikSaveTIFFJPEG(TIFF JPEG) vikSaveJPEGExif(Exif JPEG 基本 DCT) vikSaveJPEGProgressExif(Exif JPEG プログレッシブ DCT)
ImageHandle	保存対象となるイメージのメモリハンドル

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルを **FileName** プロパティに設定されたファイルに保存します。
ImageHandleが不要なメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されたメモリハンドルを **FileName** プロパティに設定されたファイルに保存します。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

その他に必要なプロパティは

FPX	**Comment**,SaveType = vikSaveFPXJpeg の場合に **JpegQuality**
GIF	**Comment**,**GifAnime**,**GifAnimeDelay**,**Interlace**,**PalBlue**,**PalGreen**,**PalRed**,**Transparent**
JPEG	**Comment**,**JpegQuality**,**JpegSubsamp**
JPEG2000	**Comment**,**JPEG2000CodeBlockHeight**,**JPEG2000CodeBlockWidth**,**JPEG2000NumResLevel**, **JPEG2000PrecinctHeight**,**JPEG2000PrecinctWidth**,**JPEG2000Reversible**,**JPEG2000Size**, **JPEG2000TileHeight**,**JPEG2000TileWidth**
PNG	**Comment**,**Interlace**,**PalBlue**,**PalGreen**,**PalRed**,**PngAlphaChannel**,**Transparent** **PngAlphaChannel** プロパティが True の場合、**ImageHandle** プロパティに RGBA の 32 ビットイメージを与えること
TIFF	**Comment**,**TiffAppend** SaveType = vikSaveTIFFNoncompressed, vikSaveTIFFPackbits, vikSaveTIFFLZW の場合に **TiffColorSpace** SaveType = vikSaveTIFFJPEG の場合に **JpegQuality** GROUP4 以外の形式で保存する場合に **TiffSaveOneStrip**
Exif	**Exif**

となります。
ベクトルイメージをラスタイメージとして保存するには、**SaveFile** メソッドを実行する前に **Vector.VectorToRaster** メソッドを実行する必要があります。また、保存処理中の進捗状況ダイアログを表示させるためにキャプション、メッセージ、ボタンをそれぞれ **Caption**,**Message**,**ButtonName** プロパティに設定することもできます。

ラスタイメージの場合には保存可能なビット数があります。対象外のビット数を保存しようとするとエラーとなりますので注意してください。下記に保存可能なビット数を示します。

形式	保存対象となるイメージのビット数
BMP(非圧縮)	1,4,8,16(グレーは除く),24,32
BMP(圧縮)	4,8
JPEG(基本 DCT)	8 ビットグレー,24
JPEG(プログレッシブ DCT)	8 ビットグレー,24
GIF	1,4,8
TIFF(非圧縮)	1,4,8,16(グレーは除く),24,32
TIFF(CCITTRLE)	1
TIFF(GROUP3-1D)	1
TIFF(GROUP3-2D)	1
TIFF(GROUP4)	1
TIFF(PACKBITS)	1,4,8,16(グレーは除く),24,32
TIFF(LZW)	1,4,8,16(グレーは除く),24,32
TIFF(JPEG)	8 ビットグレー,24
PNG	1,4,8,24
FPX(非圧縮)	8 ビットグレー,24
FPX(単色圧縮)	8 ビットグレー,24
FPX(JPEG 圧縮)	8 ビットグレー,24
PCX	1,4,8,24
JPEG2000(Part1)	8 ビットグレー,24
JPEG2000(Code Stream)	8 ビットグレー,24
Exif(基本 DCT)	8 ビットグレー,24

Exif(プログレッシブ DCT)　　8 ビットグレー,24

FPX の単色圧縮は全て同じ RGB のピクセルで構成されているイメージが対象となります。
TIFF の GROUP3-1D は MH、GROUP3-2D は MR、GROUP4 は MMR と同じ形式です。
TIFF の非圧縮、PACKBITS、LZW 形式で 16 ビットイメージを保存すると 24 ビットイメージに変換されます。

BMP ファイルを 24 ビットカラーの JPEG ファイルに変換するコード例:
(1)C++Buidler

```
VImageKit1->FileIO->FileName = "Newtone.bmp";
VImageKit1->LayerNo = -1;
if (VImageKit1->FileIO->LoadFile(vikLoadBMP) == false) return;
if (VImageKit1->GetImageType() == false) return;
if (VImageKit1->BitCount != 24)
{
  if (VImageKit1->Effect->ConvertColor(24, false, false, 0) == false) return;
}
VImageKit1->FileIO->JpegQuality = 75;
VImageKit1->FileIO->JpegSubsamp = vik411;
VImageKit1->FileIO->FileName = "Newtone.jpg";
VImageKit1->FileIO->SaveFile(vikSaveJPEG);
```

(2)Delphi

```
VImageKit1.FileIO.FileName := 'Newtone.bmp';
VImageKit1.LayerNo := -1;
if VImageKit1.FileIO.LoadFile(vikLoadBMP) = False then Exit;
if VImageKit1.GetImageType() = False then Exit;
if VImageKit1.BitCount <> 24 then
begin
  if VImageKit1.Effect.ConvertColor(24, False, False, 0) = False then Exit;
end;
VImageKit1.FileIO.JpegQuality := 75;
VImageKit1.FileIO.JpegSubsamp := vik411;
VImageKit1.FileIO.FileName := 'Newtone.jpg';
VImageKit1.FileIO.SaveFile(vikSaveJPEG);
```

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました。
ImageKit7 ActiveX は ikSaveBMP, ikSaveBMPRLE, ikSaveJPEG, ikSaveJPEGProgress, ikSaveGIF, ikSaveTIFFNoncompressed, ikSaveTIFFCcittrle, ikSaveTIFFGroup3_1D, ikSaveTIFFGroup3_2D, ikSaveTIFFGroup4, ikSaveTIFFPackbits, ikSaveTIFFLZW, ikSavePNG, ikSaveFPXNoncompressed, ikSaveFPXSingleColor, ikSaveFPXJpeg, ikSavePCX, ikSaveWMF, ikSaveEMF, ikSaveDXF, ikSaveSVG, ikSaveJPEG2000, ikSaveJPEG2000Stream, ikSaveSXFP21, ikSaveSXFSFC, ikSaveTIFFJPEG で、ImageKit8 ActiveX は vikSaveJPEGExif, vikSaveJPEGProgressExif が追加されています。

【ImageKit7 ActiveX/VCL との違い】
・FTP サーバーや HTTP(S)サーバーに直接ファイルを保存することができるようになりました。詳しくは **FileName** プロパティの説明を参照してください。
・Exif(JPEG)形式で保存できるようになりました。

SaveFileDialog(イメージキットコントロール／FileIO メソッド)

【機能】
プレビュー付きのファイルセーブダイアログを表示します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**SaveFileDialog**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**SaveFileDialog**

【引数】
ありません。

【戻り値】
保存を選択した場合は True、キャンセルの場合は False を返します。

【解説】
このメソッドを実行するには、予め **FilePath** プロパティと **FileExt** プロパティに適切な値が設定されている必要があります。
ただし、**FileName** プロパティにフルパスが設定されている場合は **FilePath** プロパティは無効です。
メソッド実行前に **ExtendedDialog** プロパティに True を設定すると、ダイアログ初期表示時にプレビューとファイル情報表示のチェックボックスが表示されます。そのため、**Preview** と **Infomation** プロパティが False でもダイアログ表示時にそれぞれ表示することができます。
保存ボタンを選択するとファイル名がフルパスで **FileName** プロパティに設定され、選択されたファイルの種類が **SaveFileDlgFileType** プロパティに設定されます。

例:
(1)C++Builder
```
bool Ret;
VImageKit1->FileIO->FileExt = "*;BMP;JPG;";
VImageKit1->FileIO->FilePath = "C:¥¥My Pictures";
VImageKit1->FileIO->FileName = "";//この場合、FilePath プロパティは有効
//FileName プロパティにフルパスでファイル名を設定すると FilePath プロパティは無効
//VImageKit1->FileIO->FileName = "C:¥¥Images¥¥001.jpg";
VImageKit1->FileIO->ExtendedDialog = true;
VImageKit1->FileIO->Preview = false;
VImageKit1->FileIO->Information = false;
Ret = VImageKit1->FileIO->SaveFileDialog();
```
(2)Delphi
```
Ret: Boolean;
VImageKit1.FileIO.FileExt := '*;BMP;JPG;';
VImageKit1.FileIO.FilePath := 'C:¥My Pictures';
VImageKit1.FileIO.FileName := '';    //この場合、FilePath プロパティは有効
//FileName プロパティにフルパスでファイル名を設定すると FilePath プロパティは無効
//VImageKit1.FileIO.FileName := 'C:¥Images¥001.jpg';
VImageKit1.FileIO.ExtendedDialog := True;
VImageKit1.FileIO.Preview := False;
VImageKit1.FileIO.Information := False;
Ret := VImageKit1.FileIO.SaveFileDialog;
```

SaveFileMem(イメージキットコントロール／FileIO メソッド)

【機能】
イメージを Raw データへ保存します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**SaveFileMem**(TVIkSaveFile SaveType, NativeUInt ImageHandle)
[*bool =*]*imagekitcontrolname*->**FileIO**->**SaveFileMem**(TVIkSaveFile SaveType)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**SaveFileMem**(SaveType: TVIkSaveFile; ImageHandle: THandle)
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**SaveFileMem**(SaveType: TVIkSaveFile)

【TVIkSaveFile 型】
ユニット
IkInit
type
TVIkSaveFile = (vikSaveBMP, vikSaveBMPRLE, vikSaveJPEG, vikSaveJPEGProgress, vikSaveGIF, vikSaveTIFFNoncompressed, vikSaveTIFFCcittrle, vikSaveTIFFGroup3_1D, vikSaveTIFFGroup3_2D, vikSaveTIFFGroup4, vikSaveTIFFPackbits, vikSaveTIFFLZW, vikSavePNG, vikSaveFPXNoncompressed, vikSaveFPXSingleColor, vikSaveFPXJpeg, vikSavePCX, vikSaveWMF, vikSaveEMF, vikSaveDXF, vikSaveSVG, vikSaveJPEG2000, vikSaveJPEG2000Stream, vikSaveSXFP21, vikSaveSXFSFC, vikSaveTIFFJPEG, vikSaveJPEGExif, vikSaveJPEGProgressExif);

【引数】

名称	内容
SaveType	vikSaveBMP(BMP 非圧縮) vikSaveBMPRLE(BMP 圧縮) vikSaveJPEG(JPEG 基本 DCT) vikSaveJPEGProgress(JPEG プログレッシブ DCT) vikSaveGIF(GIF) vikSaveTIFFNoncompressed(TIFF 非圧縮) vikSaveTIFFCcittrle(TIFF CCITTRLE) vikSaveTIFFGroup3_1D(TIFF GROUP3-1D) vikSaveTIFFGroup3_2D(TIFF GROUP3-2D) vikSaveTIFFGroup4(TIFF GROUP4) vikSaveTIFFPackbits(TIFF PACKBITS) vikSaveTIFFLZW(TIFF LZW) vikSavePNG(PNG) vikSaveFPXNoncompressed(FPX 非圧縮) vikSaveFPXSingleColor(FPX 単色圧縮) vikSaveFPXJpeg(FPX JPEG 圧縮) vikSavePCX(PCX) vikSaveWMF(WMF) vikSaveEMF(EMF) vikSaveDXF(DXF) vikSaveSVG(SVG) vikSaveJPEG2000(JPEG2000 Part1) vikSaveJPEG2000Stream(JPEG2000 Code Stream) vikSaveSXFP21(SXF p21) vikSaveSXFSFC(SXF sfc) vikSaveTIFFJPEG (TIFF JPEG) vikSaveJPEGExif(Exif JPEG 基本 DCT) vikSaveJPEGProgressExif(Exif JPEG プログレッシブ DCT)
ImageHandle	保存対象となるイメージのメモリハンドル

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数のImageHandleに有効なメモリハンドルを与えた場合
ImageHandle に設定されたメモリハンドルを **ImageHandleRawData** プロパティに Raw データとして保存します。
ImageHandleが不要なメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合
LayerNo プロパティが示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されたメモリハンドルを **ImageHandleRawData** プロパティに Raw データとして保存します。
※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

その他に必要なプロパティは

FPX	**Comment**,SaveType = vikSaveFPXJpeg の場合に **JpegQuality**
GIF	**Comment**,**GifAnime**,**GifAnimeDelay**,**Interlace**,**PalBlue**,**PalGreen**,**PalRed**,**Transparent**
JPEG	**Comment**,**JpegQuality**,**JpegSubsamp**
JPEG2000	**Comment**,**JPEG2000CodeBlockHeight**,**JPEG2000CodeBlockWidth**,**JPEG2000NumResLevel**, **JPEG2000PrecinctHeight**,**JPEG2000PrecinctWidth**,**JPEG2000Reversible**,**JPEG2000Size**, **JPEG2000TileHeight**,**JPEG2000TileWidth**
PNG	**Comment**,**Interlace**,**PalBlue**,**PalGreen**,**PalRed**,**PngAlphaChannel**,**Transparent** **PngAlphaChannel** プロパティが True の場合、**ImageHandle** プロパティに RGBA の 32 ビットイメージを与えること
TIFF	**Comment**,**TiffAppend** SaveType = vikSaveTIFFNoncompressed, vikSaveTIFFPackbits, vikSaveTIFFLZW の場合に **TiffColorSpace** SaveType = vikSaveTIFFJPEG の場合に **JpegQuality** GROUP4 以外の形式で保存する場合に **TiffSaveOneStrip**
Exif	**Exif**

となります。
ベクトルイメージをラスタイメージとして保存するには、**SaveFileMem** メソッドを実行する前に **Vector.VectorToRaster** メソッドを実行する必要があります。また、保存処理中の進捗状況ダイアログを表示させるためにキャプション、メッセージ、ボタンをそれぞれ **Caption**,**Message**,**ButtonName** プロパティに設定することもできます。
ファイルあるいは Raw データに保存する違いはありますが、動作としては **SaveFile** メソッドと同じです。
(注意)
FPX,SXF 形式の場合、内部で一時的にテンポラリファイルを作成します。

ラスタイメージの場合には保存可能なビット数があります。対象外のビット数を保存しようとするとエラーとなりますので注意してください。下記に保存可能なビット数を示します。

形式	保存対象となるイメージのビット数
BMP(非圧縮)	1,4,8,16(グレーは除く),24,32
BMP(圧縮)	4,8
JPEG(基本 DCT)	8 ビットグレー,24
JPEG(プログレッシブ DCT)	8 ビットグレー,24
GIF	1,4,8
TIFF(非圧縮)	1,4,8,16(グレーは除く),24,32
TIFF(CCITTRLE)	1
TIFF(GROUP3-1D)	1
TIFF(GROUP3-2D)	1
TIFF(GROUP4)	1
TIFF(PACKBITS)	1,4,8,16(グレーは除く),24,32
TIFF(LZW)	1,4,8,16(グレーは除く),24,32
TIFF(JPEG)	8 ビットグレー,24
PNG	1,4,8,24
FPX(非圧縮)	8 ビットグレー,24
FPX(単色圧縮)	8 ビットグレー,24
FPX(JPEG 圧縮)	8 ビットグレー,24
PCX	1,4,8,24

JPEG2000(Part1)　　8 ビットグレー,24
JPEG2000(Code Stream)　　8 ビットグレー,24
Exif(基本 DCT)　　8 ビットグレー,24
Exif(プログレッシブ DCT)　　8 ビットグレー,24

FPX の単色圧縮は全て同じ RGB のピクセルで構成されているイメージが対象となります。
TIFF の GROUP3-1D は MH、GROUP3-2D は MR、GROUP4 は MMR と同じ形式です。
TIFF の非圧縮、PACKBITS、LZW 形式で 16 ビットイメージを保存すると 24 ビットイメージに変換されます。

BMP ファイルを 24 ビットカラーの JPEG 形式の Raw データに変換するコード例:
(1)C++Buidler

```
VImageKit1->FileIO->FileName = "Newtone.bmp";
VImageKit1->LayerNo = -1;
if (VImageKit1->FileIO->LoadFile(vikLoadBMP) == false) return;
if (VImageKit1->GetImageType() == false) return;
if (VImageKit1->BitCount != 24)
{
   if (VImageKit1->Effect->ConvertColor(24, false, false, 0) == false) return;
}
VImageKit1->FileIO->JpegQuality = 75;
VImageKit1->FileIO->JpegSubsamp = vik411;
VImageKit1->FileIO->SaveFileMem(vikSaveJPEG);
```

(2)Delphi

```
VImageKit1.FileIO.FileName := 'Newtone.bmp';
VImageKit1.LayerNo := -1;
if VImageKit1.FileIO.LoadFile(vikLoadBMP) = False then Exit;
if VImageKit1.GetImageType() = False then Exit;
if VImageKit1.BitCount <> 24 then
begin
   if VImageKit1.Effect.ConvertColor(24, False, False, 0) = False then Exit;
end;
VImageKit1.FileIO.JpegQuality := 75;
VImageKit1.FileIO.JpegSubsamp := vik411;
VImageKit1.FileIO.SaveFileMem(vikSaveJPEG);
```

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました。
ImageKit7 ActiveX は ikSaveBMP, ikSaveBMPRLE, ikSaveJPEG, ikSaveJPEGProgress, ikSaveGIF, ikSaveTIFFNoncompressed, ikSaveTIFFCcittrle, ikSaveTIFFGroup3_1D, ikSaveTIFFGroup3_2D, ikSaveTIFFGroup4, ikSaveTIFFPackbits, ikSaveTIFFLZW, ikSavePNG, ikSaveFPXNoncompressed, ikSaveFPXSingleColor, ikSaveFPXJpeg, ikSavePCX, ikSaveWMF, ikSaveEMF, ikSaveDXF, ikSaveSVG, ikSaveJPEG2000, ikSaveJPEG2000Stream, ikSaveSXFP21, ikSaveSXFSFC, ikSaveTIFFJPEG で、ImageKit8 ActiveX は vikSaveJPEGExif, vikSaveJPEGProgressExif が追加されています。

【ImageKit7 ActiveX/VCL との違い】
Exif(JPEG)形式で保存できるようになりました。

SaveToStream（イメージキットコントロール／FileIO メソッド）

【機能】
イメージをストリームに保存します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**SaveToStream**(TStream* Stream)
(2)Delphi　*imagekitcontrolname*.**FileIO**.**SaveToStream**(Stream: TStream)

【引数】

名称	内容
Stream	ストリーム

【戻り値】
ありません。

【解説】
メソッドを実行すると **ImageHandleRawData** プロパティに設定された Raw データをストリームに保存します。

コード例：
(1)C++Builder

```
TMemoryStream *Stream;

Stream = new TMemoryStream();

// File -> Rawdata
VImageKit1->FileIO->FileName = "c:¥¥001.jpg"
VImageKit1->FileIO->FileLoadAsRawData();

// Rawdata -> Stream
VImageKit1->FileIO->SaveToStream(Stream);

// Clear Rawdata
VImageKit1->FileIO->ImageHandleRawData = 0;

// Stream -> Rawdata
Stream->Position = 0; //  ストリームの先頭にリセット
VImageKit1->FileIO->LoadFromStream(Stream);

// Rawdata -> File
VImageKit1->FileIO->FileName = "c:¥¥abc.jpg"
VImageKit1->FileIO->FileSaveAsRawData();

delete Stream;
```

(2)Delphi

```
Stream: TMemoryStream;

Stream := TMemoryStream.Create;

// File -> Rawdata
VImageKit1.FileIO.FileName := 'c:¥001.jpg';
VImageKit1.FileIO.FileLoadAsRawData;

// Rawdata -> Stream
VImageKit1.FileIO.SaveToStream(Stream);
```

```
// Clear Rawdata
VImageKit1.FileIO.ImageHandleRawData := 0;

// Stream -> Rawdata
Stream.Position := 0; //  ストリームの先頭にリセット
VImageKit1.FileIO.LoadFromStream(Stream);

// Rawdata -> File
VImageKit1.FileIO.FileName := 'c:¥abc.jpg';
VImageKit1.FileIO.FileSaveAsRawData;

Stream.Free;
```

【ImageKit7/8/9/10 ActiveX との違い】
戻り値がなくなり、引数が追加されました。

YCCBmpPlaneFileLoad（イメージキットコントロール／FileIO メソッド）

【機能】
YCrCb プレーン毎に保存してある BMP ファイルからラスタイメージを読み込みます。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**YCCBmpPlaneFileLoad**(const UnicodeString YFileName, const UnicodeString CrFileName, const UnicodeString CbFileName)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO**.**YCCBmpPlaneFileLoad**(const YFileName: string; const CrFileName: string; const CbFileName: string)

【引数】

名称	内容
YFileName	Y プレーンとして読み込むファイル名
CrFileName	Cr プレーンとして読み込むファイル名
CbFileName	Cb プレーンとして読み込むファイル名

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
YCrCb それぞれのプレーンを表すイメージは 8 ビットグレーで、出力イメージは 24 ビットカラーです。
成功すると **LayerNo** プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）にイメージのメモリハンドルが設定されます。

YCCBmpPlaneFileSave（イメージキットコントロール／FileIO メソッド）

【機能】

ラスタイメージを YCrCb プレーン毎に BMP 形式でファイルに保存します。

【書式】

(1)C++Builder

[*bool =*]*imagekitcontrolname*->**FileIO**->**YCCBmpPlaneFileSave**(const UnicodeString YFileName, const UnicodeString CrFileName, const UnicodeString CbFileName, NativeUInt ImageHandle)

[*bool =*]*imagekitcontrolname*->**FileIO**->**YCCBmpPlaneFileSave**(const UnicodeString YFileName, const UnicodeString CrFileName, const UnicodeString CbFileName)

(2)Delphi

[*Boolean =*]*imagekitcontrolname*.**FileIO**.**YCCBmpPlaneFileSave**(const YFileName: string; const CrFileName: string; const CbFileName: string; ImageHandle: THandle)

[*Boolean =*]*imagekitcontrolname*.**FileIO**.**YCCBmpPlaneFileSave**(const YFileName: string; const CrFileName: string; const CbFileName: string)

【引数】

名称	内容
YFileName	Y プレーンとして保存するファイル名
CrFileName	Cr プレーンとして保存するファイル名
CbFileName	Cb プレーンとして保存するファイル名
ImageHandle	保存対象となるイメージのメモリハンドル

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

引数のImageHandleに有効なメモリハンドルを与えた場合

ImageHandle に設定されたメモリハンドルを YCrCb のそれぞれのプレーン毎にファイルに保存します。

ImageHandleが不要なメソッドを使用、もしくは引数のImageHandleに 0 を与えた場合

LayerNo プロパティが示すプロパティ（**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ）に設定されたメモリハンドルを YCrCb のそれぞれのプレーン毎にファイルに保存します。

※LayerNo=-1 であれば ImageHandle、LayerNo=0～99 であれば Layer[LayerNo].ImageHandle

保存対象イメージは 24 ビットカラーで、保存されるイメージは 8 ビットグレーとなります。

Exif(イメージキットコントロール／カスタム階層プロパティ)

【機能】
Exif(JPEG)ファイルから主画像とサムネイル画像の情報を取得します。Exif はバージョン 2.3 の一部のタグにも対応しています。

【情報の取得について】
FileIO.GetImageFileType(Mem)メソッドの実行により、すべての項目が取得されるわけではなく、機器や画像の特性により設定されない項目もあります。
詳しくは、「社団法人　日本電子工業振興協会」発行の「JEIDA 規格　デジタルスチルカメラ用画像ファイルフォーマット規格(Exif)」を参照してください。

【画像情報の構成】
画像情報
　|
　├──── 主画像情報
　|
　├──── Exif 情報
　|
　├──── GPS 情報
　|
　└──── サムネイル画像情報

●プロパティ一覧(アルファベット順)

カスタムプロパティ	内容
Altitude0	高度の分子
Altitude1	高度の分母
AltitudeRef	高度の基準
ApertureValue0	レンズの絞り値の分子
ApertureValue1	レンズの絞り値の分母
BrightnessValue0	輝度値の分子
BrightnessValue1	輝度値の分母
ColorSpace	色空間を示す情報
ComponentsConfiguration	各コンポーネントのチャネルを第 1 コンポーネントから第 4 コンポーネントの順に示す
CompressedBitsPerPixel0	画像圧縮時に設定された圧縮モードの分子
CompressedBitsPerPixel1	画像圧縮時に設定された圧縮モードの分母
Contrast	撮影コントラスト
DateStamp	GPS 日付
DateTimeDigitized	画像がデジタルデータ化された日付と時間
DateTimeOriginal	原画像データの生成された日付と時間
ExposureBiasValue0	露光補正値の分子
ExposureBiasValue1	露光補正値の分母
ExposureIndex0	画像が取り込まれた時、カメラまたは入力機器が選択した露出のインデックスの分子
ExposureIndex1	画像が取り込まれた時、カメラまたは入力機器が選択した露出のインデックスの分母
ExposureProgram	撮影時にカメラが使用した露出プログラムのクラス
ExposureTime0	露出時間の分子
ExposureTime1	露出時間の分母
FileSource	画像のソース
Flash	フラッシュ
FlashEnergy0	画像が取り込まれた時に使用されたストロボのエネルギーの分子
FlashEnergy1	画像が取り込まれた時に使用されたストロボのエネルギーの分母
FlashPixVersion	FPXR ファイルの FlashPix フォーマットへの対応バージョン
FNumber0	F ナンバーの分子
FNumber1	F ナンバーの分母
FocalLength0	レンズ焦点距離の分子
FocalLength1	レンズ焦点距離の分母

FocalPlaneResolutionUnit	焦点面解像度単位
FocalPlaneXResolution0	焦点面の幅の解像度の分子
FocalPlaneXResolution1	焦点面の幅の解像度の分母
FocalPlaneYResolution0	焦点面の高さの解像度の分子
FocalPlaneYResolution1	焦点面の高さの解像度の分母
GPSVersion	GPS タグのバージョン
InteroperabilityIndex	互換性識別子
InteroperabilityVersion	互換性バージョン
IsExif	Exif かどうかを示す
ISOSpeedRatings	ISO 12232xiv で規定されるカメラまたは入力機器の ISO Speed および ISO Latitude
Latitude	緯度の方角
LatitudeD0	緯度-度の分子
LatitudeD1	緯度-度の分母
LatitudeM0	緯度-分の分子
LatitudeM1	緯度-分の分母
LatitudeS0	緯度-秒の分子
LatitudeS1	緯度-秒の分母
LightSource	光源の種類
Longitude	経度の方角
LongitudeD0	経度-度の分子
LongitudeD1	経度-度の分母
LongitudeM0	経度-分の分子
LongitudeM1	経度-分の分母
LongitudeS0	経度-秒の分子
LongitudeS1	経度-秒の分母
MainArtist	作者名（主画像）
MainCopyright	撮影著作権者/編集著作権者（主画像）
MainDateTime	ファイル作成日時（主画像）
MainImageDescription	画像タイトル（主画像）
MainMake	画像入力機器のメーカ名（主画像）
MainModel	画像入力機器のモデル名（主画像）
MainOrientation	行と列の観点から見た画像の方向（主画像）
MainResolutionUnit	解像度の単位（主画像）
MainSoftware	使用ソフトウェア名（主画像）
MainXResolution0	イメージの横方向の 1MainResolutionUnit あたりの画素数（主画像）の分子
MainXResolution1	イメージの横方向の 1MainResolutionUnit あたりの画素数（主画像）の分母
MainYCbCrPositioning	輝度サンプルに対するクロマサンプルの相対的配置を特定（主画像）
MainYResolution0	イメージの縦方向の 1MainResolutionUnit あたりの画素数（主画像）の分子
MainYResolution1	イメージの縦方向の 1MainResolutionUnit あたりの画素数（主画像）の分母
MakerNote	メーカノート
MapDatum	測位に用いた地図データ
MaxApertureValue0	レンズの最小 F 値の分子
MaxApertureValue1	レンズの最小 F 値の分母
MeteringMode	測光方式
PixelXDimension	実効画像幅
PixelYDimension	実効画像高さ
ProcessingMethod	測位方式の名称
ProcessingMethodID	ProcessingMethod に書かれる文字コードを判別するための識別コード
Saturation	撮影彩度
SceneCaptureType	撮影シーンタイプ
SceneType	画像のシーンのタイプ
SensingMethod	カメラまたは入力機器で使用される画像センサのタイプ
Sharpness	撮影シャープネス
ShutterSpeedValue0	シャッタースピードの分子
ShutterSpeedValue1	シャッタースピードの分母
SpectralSensitivity	撮影に用いたカメラの各チャネルのスペクトル感度
SubjectDistance0	被写体距離の分子
SubjectDistance1	被写体距離の分母
SubjectLocationX	主要被写体のおおよその X 座標値

SubjectLocationY	主要被写体のおおよそのＹ座標値
SubSecTime	MainDateTime に関連して時刻を小数点以下の秒単位まで記録
SubSecTimeDigitized	DateTimeDigitized に関連して時刻を小数点以下の秒単位まで記録
SubSecTimeOriginal	DateTimeOriginal に関連して時刻を小数点以下の秒単位まで記録
ThumbArtist	作者名（サムネイル画像）
ThumbCompression	サムネイル画像の圧縮方法
ThumbCopyright	撮影著作権者/編集著作権者（サムネイル画像）
ThumbDateTime	ファイル作成日時（サムネイル画像）
ThumbImageDescription	画像タイトル（サムネイル画像）
ThumbImageHandle	サムネイル画像のメモリハンドル
ThumbMake	画像入力機器のメーカ名（サムネイル画像）
ThumbModel	画像入力機器のモデル名（サムネイル画像）
ThumbOrientation	行と列の観点から見た画像の方向（サムネイル画像）
ThumbResolutionUnit	解像度の単位（サムネイル画像）
ThumbSoftware	使用ソフトウェア名（サムネイル画像）
ThumbXResolution0	イメージの横方向の 1ThumbResolutionUnit あたりの画素数（サムネイル画像）の分子
ThumbXResolution1	イメージの横方向の 1ThumbResolutionUnit あたりの画素数（サムネイル画像）の分母
ThumbYCbCrPositioning	輝度サンプルに対するクロマサンプルの相対的配置を特定（サムネイル画像）
ThumbYResolution0	イメージの縦方向の 1ThumbResolutionUnit あたりの画素数（サムネイル画像）の分子
ThumbYResolution1	イメージの縦方向の 1ThumbResolutionUnit あたりの画素数（サムネイル画像）の分母
TimeStampH0	GPS 時間-時間の分子
TimeStampH1	GPS 時間-時間の分母
TimeStampM0	GPS 時間-分の分子
TimeStampM1	GPS 時間-分の分母
TimeStampS0	GPS 時間-秒の分子
TimeStampS1	GPS 時間-秒の分母
UserComment	画像に対して Exif ユーザがキーワードやコメントを書き込むためのタグ
UserCommentID	UserComment に書かれる文字コードを判別するための識別コード
Version	Exif の対応バージョン
WhiteBalance	ホワイトバランス

Altitude0,Altitude1,LatitudeD0,LatitudeD1,LatitudeM0,LatitudeM1,LatitudeS0,LatitudeS1, LongitudeD0,LongitudeD1,LongitudeM0,LongitudeM1,LongitudeS0,LongitudeS1 (イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※Altitude0 にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->Altitude0 [= *int*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.Altitude0 [= *Integer*]

【設定値】
Altitude0 は、高度(m)の分子
Altitude1 は、高度(m)の分母
高度 = Altitude0 / Altitude1

LatitudeD0 は、緯度-度の分子
LatitudeD1 は、緯度-度の分母
緯度-度 = LatitudeD0 / LatitudeD1

LatitudeM0 は、緯度-分の分子
LatitudeM1 は、緯度-分の分母
緯度-分 = LatitudeM0 / LatitudeM1

LatitudeS0 は、緯度-秒の分子
LatitudeS1 は、緯度-秒の分母
緯度-秒 = LatitudeS0 / LatitudeS1

LongitudeD0 は、経度-度の分子
LongitudeD1 は、経度-度の分母
経度-度 = LongitudeD0 / LongitudeD1

LongitudeM0 は、経度-分の分子
LongitudeM1 は、経度-分の分母
経度-分 = LongitudeM0 / LongitudeM1

LongitudeS0 は、経度-秒の分子
LongitudeS1 は、経度-秒の分母
経度-秒 = LongitudeS0 / LongitudeS1

【解説】
Exif の GPS に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

AltitudeRef(イメージキットコントロール／Exif プロパティ)

【機能】
高度の基準を示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**AltitudeRef** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**Exif**.**AltitudeRef** [= *Smallint*]

【設定値】
高度の基準
基準が海抜であり高度が海抜よりも高い場合は 0、高度が海抜よりも低い場合は 1。基準単位はメートル。

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の GPS に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

ApertureValue0,ApertureValue1,BrightnessValue0,BrightnessValue1,ExposureBiasValue0, ExposureBiasValue1,MaxApertureValue0,MaxApertureValue1,ShutterSpeedValue0, ShutterSpeedValue1(イメージキットコントロール／Exifプロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※ApertureValue0 にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->ApertureValue0 [*= int*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.ApertureValue0 [*= Integer*]

【設定値】
ApertureValue0 は、レンズの絞り値の分子
ApertureValue1 は、レンズの絞り値の分母
レンズの絞り値(単位は APEX 値) = ApertureValue0 / ApertureValue1

BrightnessValue0 は、輝度値の分子
BrightnessValue1 は、輝度値の分母
輝度値(単位は APEX 値) = BrightnessValue0 / BrightnessValue1
一般的な記載範囲は-99.99 から 99.99 です。

ExposureBiasValue0 は、露光補正値の分子
ExposureBiasValue1 は、露光補正値の分母
露光補正値(単位は APEX 値) = ExposureBiasValue0 / ExposureBiasValue1
一般的な記載範囲は-99.99 から 99.99 です。

MaxApertureValue0 は、レンズの最小 F 値の分子
MaxApertureValue1 は、レンズの最小 F 値の分母
レンズの最小 F 値(単位は APEX 値) = MaxApertureValue0 / MaxApertureValue1
一般的な記載範囲は 00.00 から 99.99 です。

ShutterSpeedValue0 は、シャッタースピードの分子
ShutterSpeedValue1 は、シャッタースピードの分母
シャッタースピード(単位は APEX 値) = ShutterSpeedValue0 / ShutterSpeedValue1

※APEX は Additive System of Photographic Exposure の略

【解説】
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

ColorSpace(イメージキットコントロール／Exif プロパティ)

【機能】
色空間を示す情報です。

【書式】
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->ColorSpace [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.ColorSpace [= *Smallint*]

【設定値】

値	説明
1	sRGB

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の画像データの特性に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

ComponentsConfiguration(イメージキットコントロール／Exif プロパティ)

【機能】
各コンポーネントのチャネルを第 1 コンポーネントから第 4 コンポーネントの順に示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->Exif->ComponentsConfiguration** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.ComponentsConfiguration** [= *string*]

【設定値】

値	説明
0	存在しない
1	Y
2	Cb
3	Cr
4	R
5	G
6	B

RGB の時は 4560 で、JPEG の時は 1230 となります。

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の画像データの構成に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

CompressedBitsPerPixel0,CompressedBitsPerPixel1（イメージキットコントロール／Exif プロパティ）

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※CompressedBitsPerPixel0 にて説明（CompressedBitsPerPixel1 も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->CompressedBitsPerPixel0 [= *int*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.CompressedBitsPerPixel0 [= *Integer*]

【設定値】
CompressedBitsPerPixel0 は、画像圧縮時に設定された圧縮モードの分子
CompressedBitsPerPixel1 は、画像圧縮時に設定された圧縮モードの分母
画像圧縮時に設定された圧縮モード（単位は bit/pixel） = CompressedBitsPerPixel0 / CompressedBitsPerPixel1

【解説】
Exif の画像データの構成に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

Contrast,Saturation,Sharpness（イメージキットコントロール／Exif プロパティ）

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。

【書式】
※Contrast にて説明（その他も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->Contrast [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.Contrast [= *Smallint*]

【設定値】
Contrast は、撮影時にカメラが画像に施したコントラスト処理傾向
Saturation は、撮影時にカメラが画像に施した彩度処理傾向
Sharpness は、撮影時にカメラが画像に施したシャープネス処理傾向

【解説】
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

DateStamp(イメージキットコントロール／Exifプロパティ)

【機能】
GPS 日付を示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**DateStamp** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.DateStamp** [= *string*]

【設定値】
GPS 日付
UTC(Coordinated Universal Time)に基づく日付情報。フォーマットは"YYYY:MM:DD"。

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の GPS に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

DateTimeDigitized,DateTimeOriginal(イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※DateTimeDigitized にて説明(DateTimeOriginal も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->DateTimeDigitized [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.DateTimeDigitized [= *string*]

【設定値】
DateTimeDigitized は、画像がデジタルデータ化された日付と時間
DateTimeOriginal は、原画像データの生成された日付と時間(デジタルカメラでは撮影された日付と時間)

フォーマットは"YYYY:MM:DD HH:MM:SS"。時間は 24 時間表示とし、日付と時間の間に空白文字を 1 つ挿入。
例: "1996:09:01 09:15:30"

【解説】
Exif の日時に関するタグです。
デジタルカメラなどで撮影され、同時にファイルが記録される場合は DateTimeDigitized と DateTimeOriginal は同じになります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

ExposureIndex0,ExposureIndex1,ExposureTime0,ExposureTime1,FlashEnergy0,FlashEnergy1, FNumber0,FNumber1,FocalLength0,FocalLength1,SubjectDistance0,SubjectDistance1 (イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※ExposureIndex0 にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->ExposureIndex0 [*= int*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.ExposureIndex0 [*= Integer*]

【設定値】
ExposureIndex0 は、画像が取り込まれた時、カメラまたは入力機器が選択した露出のインデックスの分子
ExposureIndex1 は、画像が取り込まれた時、カメラまたは入力機器が選択した露出のインデックスの分母
露出インデックス = ExposureIndex0 / ExposureIndex1

ExposureTime0 は、露出時間の分子
ExposureTime1 は、露出時間の分母
露出時間(単位は秒) = ExposureTime0 / ExposureTime1

FlashEnergy0 は、画像が取り込まれた時に使用されたストロボのエネルギーの分子
FlashEnergy1 は、画像が取り込まれた時に使用されたストロボのエネルギーの分母
画像が取り込まれた時に使用されたストロボのエネルギー(測定単位は BCPS) = FlashEnergy0 / FlashEnergy1
※BCPS は Beam Candle Power Seconds の略

FNumber0 は、F ナンバーの分子
FNumber1 は、F ナンバーの分母
F ナンバー = FNumber0 / FNumber1

FocalLength0 は、撮影レンズの実焦点距離の分子
FocalLength1 は、撮影レンズの実焦点距離の分母
撮影レンズの実焦点距離(単位は mm) = FocalLength0 / FocalLength1

SubjectDistance0 は、被写体距離の分子
SubjectDistance1 は、被写体距離の分母
被写体距離(単位は m) = SubjectDistance0 / SubjectDistance1

【解説】
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

ExposureProgram(イメージキットコントロール／Exif プロパティ)

【機能】
撮影時にカメラが使用した露出プログラムのクラスを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**ExposureProgram** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.ExposureProgram** [= *Smallint*]

【設定値】

値	説明
0	未定義
1	マニュアル
2	ノーマルプログラム
3	絞り優先
4	シャッター優先
5	creative プログラム(被写界深度方向にバイアス)
6	action プログラム(シャッタースピード高速側にバイアス)
7	ポートレイトモード(クローズアップ撮影、背景はフォーカス外す)
8	ランドスケープモード(landscape 撮影、背景はフォーカス合う)

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

FileSource,SceneType（イメージキットコントロール／Exif プロパティ）

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※FileSource にて説明（SceneType も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->FileSource [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.FileSource [= *Smallint*]

【設定値】
FileSource は、画像のソース（デジタルカメラで記録する場合には常に 3 でなければならない）

値	説明
0	その他
1	透過型スキャナ
2	反射型スキャナ
3	デジタルカメラ

SceneType は、画像のシーンのタイプ（デジタルカメラで記録する場合には常に 1 でなければならない）

【解説】
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

Flash(イメージキットコントロール／Exif プロパティ)

【機能】
撮影時のストロボの状態を示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**Flash** [*= short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.Flash** [*= Smallint*]

【設定値】
ストロボを使用して画像が取り込まれた時に記録される値です。

ビット 0 はストロボの状態、ビット 1 および 2 はストロボのリターン状態、ビット 3 および 4 はカメラのストロボモード、ビット 5 はストロボ機能の有無、ビット 6 は赤目モードを表します。
ストロボ発光状態のビットの値(bit0)
0b = ストロボ発光せず
1b = ストロボ発光
ストロボのリターン状態の値(bit1,2)
00b = ストロボのリターン検出機能なし
01b = 予約
10b = ストロボのリターン検出されず
11b = ストロボのリターン検出
カメラのストロボモードの値(bit3,4)
00b = モード不明
01b = 強制発行モード
10b = 強制非発行モード
11b = 自動発行モード
ストロボ機能の有無(bit5)
0b = ストロボ機能有り
1b = ストロボ機能無し
カメラの赤目モードの値(bit6)
0b = 赤目軽減無しまたは不明
1b = 赤目軽減有り

「16 進表記」
0x0000 = ストロボ発光せず
0x0001 = ストロボ発光
0x0005 = ストロボ発光、リターン検出されず
0x0007 = ストロボ発光、リターン検出
※Delphi は 0x を$に置き換えてください。

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

FlashPixVersion,GPSVersion,Version(イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※FlashPixVersion にて説明(GPSVersion,Version も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->FlashPixVersion [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.FlashPixVersion [= *string*]

【設定値】
FlashPixVersion は、FPXR ファイルの FlashPix フォーマットへの対応バージョン(例:"1000")
GPSVersion は、GPS タグのバージョン(例:"2000")
Version は、Exif の対応バージョン(例:"0210")

【解説】
Exif のバージョンに関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

FocalPlaneResolutionUnit,MainResolutionUnit,ThumbResolutionUnit(イメージキットコントロール／Exifプロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※FocalPlaneResolutionUnit にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->FocalPlaneResolutionUnit [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.FocalPlaneResolutionUnit [= *Smallint*]

【設定値】
FocalPlaneResolutionUnit は、焦点面の幅の解像度(FocalPlaneXResolution0/FocalPlaneXResolution1)と焦点面の高さの解像度(FocalPlaneYResolution0/FocalPlaneYResolution1)に対する測定単位を表します。

MainResolutionUnit は、MainXResolution0/MainXResolution1 と MainYResolution0/MainYResolution1 に対する解像度の単位を表します。

ThumbResolutionUnit は、ThumbXResolution0/ThumbXResolution1 と ThumbYResolution0/ThumbYResolution1 に対する解像度の単位を表します。

値	説明
2	インチ
3	センチメートル

【解説】
FocalPlaneResolutionUnit は Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

FocalPlaneXResolution0,FocalPlaneXResolution1,FocalPlaneYResolution0, FocalPlaneYResolution1 (イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※FocalPlaneXResolution0 にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->FocalPlaneXResolution0 [= *int*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.FocalPlaneXResolution0 [= *Integer*]

【設定値】
FocalPlaneXResolution0 は、焦点面の幅の解像度の分子
FocalPlaneXResolution1 は、焦点面の幅の解像度の分母
焦点面の幅の解像度(単位は FocalPlaneResolutionUnit 値) = FocalPlaneXResolution0 / FocalPlaneXResolution1
カメラのフォーカルプレーン上での FocalPlaneResolutionUnit あたりの画像幅(X)方向の画素数を表します。

FocalPlaneYResolution0 は、焦点面の高さの解像度の分子
FocalPlaneYResolution1 は、焦点面の高さの解像度の分母
焦点面の高さの解像度(単位は FocalPlaneResolutionUnit 値) = FocalPlaneYResolution0 / FocalPlaneYResolution1
カメラのフォーカルプレーン上での FocalPlaneResolutionUnit あたりの画像高さ(Y)方向の画素数を表します。

【解説】
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

InteroperabilityIndex,InteroperabilityVersion(イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※InteroperabilityIndex にて説明(InteroperabilityVersion も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->InteroperabilityIndex [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.InteroperabilityIndex [= *string*]

【設定値】
InteroperabilityIndex
互換性の規則の種類。

"R98"　ExifR98 で規定される R98 ファイルおよび Design rule for Camera File system で規定される DCF 基本ファイル
"THM"　Design rule for Camera File system で規定される DCF サムネイルファイル
"R03"　Design rule for Camera File system で規定される DCF オプションファイル

InteroperabilityVersion
互換性のバージョン(例:"0100")。

【解説】
Exif の互換性に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

IsExif(イメージキットコントロール／Exif プロパティ)

【機能】
ファイル形式が Exif かどうかを取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**IsExif** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.IsExif** [= *Boolean*]

【参照値】

値	説明
True	Exif
False	Exif 以外

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。

【値の設定】 不可

【値の参照】 実行時

ISOSpeedRatings(イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。

【書式】
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->ISOSpeedRatings [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.ISOSpeedRatings [= *Smallint*]

【設定値】
ISO 12232xiv で規定されるカメラまたは入力機器の ISO Speed および ISO Latitude

【解説】
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

Latitude,Longitude,MapDatum（イメージキットコントロール／Exif プロパティ）

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※Latitude にて説明（その他も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->Latitude [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.Latitude [= *string*]

【設定値】
Latitude は、緯度の方角（"N":北緯, "S":南緯）
Longitude は、経度の方角（"E":東経, "W":西経）
MapDatum は、測位に用いた地図データ。GPS 受信機が使用した測地系を示す。日本で採用されている測地系として、'TOKYO'もしくは'WGS-84'などがある。

【解説】
Exif の GPS に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

LightSource(イメージキットコントロール／Exif プロパティ)

【機能】
光源の種類を示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**LightSource** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**Exif**.**LightSource** [= *Smallint*]

【設定値】

値	説明
0	不明
1	昼光
2	蛍光灯
3	タングステン(白熱灯)
4	フラッシュ
9	晴天
10	曇天
11	日陰
12	昼光色蛍光灯(D:5700 – 7100K)
13	昼白色蛍光灯(N:4600 – 5500K)
14	白色蛍光灯(W:3800 – 4500K)
15	温白色蛍光灯(WW:3250 – 3800K)
16	電球色蛍光灯(L:2600 – 3250K)
17	標準光 A
18	標準光 B
19	標準光 C
20	D55
21	D65
22	D75
23	D50
24	ISO studio tungsten
255	その他

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

MainArtist,MainCopyright,MainDateTime,MainImageDescription,MainMake,MainModel, MainSoftware,ThumbArtist,ThumbCopyright,ThumbDateTime,ThumbImageDescription, ThumbMake,ThumbModel,ThumbSoftware(イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※MainArtist にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->MainArtist [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.MainArtist [= *string*]

【設定値】
1.主画像に関する情報
MainArtist は、主画像の作者名
MainCopyright は、主画像の撮影著作権者/編集著作権者

MainDateTime は、ファイル作成日時
フォーマットは"YYYY:MM:DD HH:MM:SS"。時間は 24 時間表示とし、日付と時間の間に空白文字を 1 つ挿入。
例: "1996:09:01 09:15:30"

MainImageDescription は、画像タイトル
MainMake は、画像入力機器のメーカ名
MainModel は、画像入力機器のモデル名
MainSoftware は、使用ソフトウェア名

2.サムネイル画像に関する情報
ThumbArtist は、サムネイル画像の作者名
ThumbCopyright は、サムネイル画像の撮影著作権者/編集著作権者

ThumbDateTime は、ファイル作成日時
フォーマットは"YYYY:MM:DD HH:MM:SS"。時間は 24 時間表示とし、日付と時間の間に空白文字を 1 つ挿入。
例: "1996:09:01 09:15:30"

ThumbImageDescription は、画像タイトル
ThumbMake は、画像入力機器のメーカ名
ThumbModel は、画像入力機器のモデル名
ThumbSoftware は、使用ソフトウェア名

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

MainOrientation,ThumbOrientation(イメージキットコントロール／Exif プロパティ)

【機能】
行と列の観点から見た画像の方向を示します。

【書式】
※MainOrientation にて説明(ThumbOrientation も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->MainOrientation [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.MainOrientation [= *Smallint*]

【設定値】

値	説明
1	0 番目の行が目で見たときの画像の上、0 番目の列が左側となる
2	0 番目の行が目で見たときの画像の上、0 番目の列が右側となる
3	0 番目の行が目で見たときの画像の下、0 番目の列が右側となる
4	0 番目の行が目で見たときの画像の下、0 番目の列が左側となる
5	0 番目の行が目で見たときの画像の左側、0 番目の列が上となる
6	0 番目の行が目で見たときの画像の右側、0 番目の列が上となる
7	0 番目の行が目で見たときの画像の右側、0 番目の列が下となる
8	0 番目の行が目で見たときの画像の左側、0 番目の列が下となる

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。
MainOrientation は主画像に関する情報で、ThumbOrientation はサムネイル画像に関する情報です。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

MainXResolution0,MainXResolution1,MainYResolution0,MainYResolution1, ThumbXResolution0,ThumbXResolution1,ThumbYResolution0,ThumbYResolution1 (イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※MainXResolution0 にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->MainXResolution0 [= *int*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.MainXResolution0 [= *Integer*]

【設定値】
1.主画像に関する情報
MainXResolution0 は、イメージの横方向の 1MainResolutionUnit あたりの画素数の分子
MainXResolution1 は、イメージの横方向の 1MainResolutionUnit あたりの画素数の分母
横方向の解像度(単位は MainResolutionUnit 値) = MainXResolution0 / MainXResolution1

MainYResolution0 は、イメージの縦方向の 1MainResolutionUnit あたりの画素数の分子
MainYResolution1 は、イメージの縦方向の 1MainResolutionUnit あたりの画素数の分母
縦方向の解像度(単位は MainResolutionUnit 値) = MainYResolution0 / MainYResolution1

2.サムネイル画像に関する情報
ThumbXResolution0 は、イメージの横方向の 1ThumbResolutionUnit あたりの画素数の分子
ThumbXResolution1 は、イメージの横方向の 1ThumbResolutionUnit あたりの画素数の分母
横方向の解像度(単位は ThumbResolutionUnit 値) = ThumbXResolution0 / ThumbXResolution1

ThumbYResolution0 は、イメージの縦方向の 1ThumbResolutionUnit あたりの画素数の分子
ThumbYResolution1 は、イメージの縦方向の 1ThumbResolutionUnit あたりの画素数の分母
縦方向の解像度(単位は ThumbResolutionUnit 値) = ThumbYResolution0 / ThumbYResolution1

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

MainYCbCrPositioning,ThumbYCbCrPositioning(イメージキットコントロール／Exif プロパティ)

【機能】
輝度サンプルに対するクロマサンプルの相対的配置を特定します。

【書式】
※MainYCbCrPositioning にて説明(ThumbYCbCrPositioning も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->MainYCbCrPositioning [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.MainYCbCrPositioning [= *Smallint*]

【設定値】

値	説明
1	中心
2	一致

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。
MainYCbCrPositioning は主画像に関する情報で、ThumbYCbCrPositioning はサムネイル画像に関する情報です。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

MakerNote,UserComment,UserCommentID（イメージキットコントロール／Exif プロパティ）

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※MakerNote にて説明（その他も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->MakerNote [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.MakerNote [= *string*]

【設定値】
MakerNote は、Exif ライターのメーカが個別の情報を記入するタグ
UserComment は、画像に対して Exif ユーザがキーワードやコメントを書き込むためのタグ

UserCommentID は、UserComment に書かれる文字コードを判別するために、識別コードを 8 バイト固定で記入し、余った領域には NULL（0x00）でパディングする。

文字コード	コード記入方法（8 バイト）	リファレンス
ASCII	0x41,0x53,0x43,0x49,0x49,0x00,0x00,0x00	ITU-T T.50 IA5
JIS	0x4A,0x49,0x53,0x00,0x00,0x00,0x00,0x00	JIS X0208-1990
Unicode	0x55,0x4E,0x49,0x43,0x4F,0x44,0x45,0x00	Unicode Standard
Undefined	0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00	Undefined

※16 進表記のため、Delphi は 0x を$に置き換えてください。

【解説】
Exif のユーザ情報に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

MeteringMode(イメージキットコントロール／Exif プロパティ)

【機能】
測光方式を示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**MeteringMode** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.MeteringMode** [= *Smallint*]

【設定値】

値	説明
0	不明
1	平均
2	中央重点
3	スポット
4	マルチスポット
5	分割測光
6	部分測光
255	その他

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

PixelXDimension,PixelYDimension（イメージキットコントロール／Exif プロパティ）

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※PixelXDimension にて説明（PixelYDimension も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->PixelXDimension [= *int*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.PixelXDimension [= *Integer*]

【設定値】
PixelXDimension は、実効画像幅
PixelYDimension は、実効画像高さ

【解説】
Exif の画像データの構成に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

ProcessingMethod,ProcessingMethodID（イメージキットコントロール／Exif プロパティ）

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※ProcessingMethod にて説明（ProcessingMethodID も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->MakerNote [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.MakerNote [= *string*]

【設定値】
ProcessingMethod は、測位に使用した方式の名称（例：″ CELLID″,″ WLAN″,″ GPS″など）

ProcessingMethodID は、ProcessingMethod に書かれる文字コードを判別するために、識別コードを 8 バイト固定で記入し、余った領域には NULL（0x00）でパディングする。

文字コード	コード記入方法（8 バイト）	リファレンス
ASCII	0x41,0x53,0x43,0x49,0x49,0x00,0x00,0x00	ITU-T T.50 IA5
JIS	0x4A,0x49,0x53,0x00,0x00,0x00,0x00,0x00	JIS X0208-1990
Unicode	0x55,0x4E,0x49,0x43,0x4F,0x44,0x45,0x00	Unicode Standard
Undefined	0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00	Undefined

※16 進表記のため、Delphi は 0x を$に置き換えてください。

【解説】
Exif の GPS 情報に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

SceneCaptureType(イメージキットコントロール／Exif プロパティ)

【機能】
撮影時の被写体種別を示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->Exif->SceneCaptureType** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.SceneCaptureType** [= *Smallint*]

【設定値】

値	説明
0	標準
1	風景
2	人物
3	夜景

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

SensingMethod(イメージキットコントロール／Exif プロパティ)

【機能】
カメラまたは入力機器で使用される画像センサのタイプを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**SensingMethod** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.SensingMethod** [= *Smallint*]

【設定値】

値	説明
1	未定義
2	単板カラーセンサ
3	2 板カラーセンサ
4	3 板カラーセンサ
5	色順次カラーセンサ
7	3 線リニアセンサ
8	色順次リニアセンサ

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

SpectralSensitivity（イメージキットコントロール／Exif プロパティ）

【機能】
撮影に用いたカメラの各チャネルのスペクトル感度を表します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**SpectralSensitivity** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.SpectralSensitivity** [= *string*]

【設定値】
ASTM Technical committee で開発された規格と互換性がある ASCII 文字列

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

SubjectLocationX,SubjectLocationY（イメージキットコントロール／Exif プロパティ）

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※SubjectLocationX にて説明（SubjectLocationY も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->SubjectLocationX [= *short*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.SubjectLocationX [= *Smallint*]

【設定値】
SubjectLocationX は、主要被写体のおおよその X 座標値
SubjectLocationY は、主要被写体のおおよその Y 座標値

【解説】
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

SubSecTime,SubSecTimeDigitized,SubSecTimeOriginal(イメージキットコントロール／Exif プロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※SubSecTime にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->SubSecTime [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.SubSecTime [= *string*]

【設定値】
SubSecTime は、MainDateTime に関連して時刻を小数点以下の秒単位までを表します。
SubSecTimeDigitized は、DateTimeDigitized に関連して時刻を小数点以下の秒単位までを表します。
SubSecTimeOriginal は、DateTimeOriginal に関連して時刻を小数点以下の秒単位までを表します。

サブセックデータが 0.130 秒の場合の例:
SubSecTime = "130"

【解説】
Exif の日時に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

ThumbCompression(イメージキットコントロール／Exif プロパティ)

【機能】
サムネイル画像データに使用された圧縮方法を示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->Exif->ThumbCompression** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.ThumbCompression** [= *Smallint*]

【設定値】

値	説明
1	非圧縮
6	JPEG 圧縮

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
サムネイル画像が JPEG 圧縮データの場合は 6 になります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
値の設定が可能になりました。

ThumbImageHandle（イメージキットコントロール／Exif プロパティ）

【機能】
サムネイル画像のメモリハンドルを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->ThumbImageHandle [= *NativeUInt*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.ThumbImageHandle [= *THandle*]

【参照値】
サムネイル画像のメモリハンドル

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。

【値の設定】 不可

【値の参照】 実行時

TimeStampH0,TimeStampH1,TimeStampM0,TimeStampM1,TimeStampS0,TimeStampS1(イメージキットコントロール／Exifプロパティ)

【機能】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、各プロパティに値が設定されます。

【書式】
※TimeStampH0 にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->FileIO->Exif->TimeStampH0 [= *int*]
(2)Delphi　*imagekitcontrolname*.FileIO.Exif.TimeStampH0 [= *Integer*]

【設定値】
TimeStampH0 は、GPS 時間-時間の分子
TimeStampH1 は、GPS 時間-時間の分母
GPS 時間-時間 = TimeStampH0 / TimeStampH1

TimeStampM0 は、GPS 時間-分の分子
TimeStampM1 は、GPS 時間-分の分母
GPS 時間-分 = TimeStampM0 / TimeStampM1

TimeStampS0 は、GPS 時間-秒の分子
TimeStampS1 は、GPS 時間-秒の分母
GPS 時間-秒= TimeStampS0 / TimeStampS1

【解説】
Exif の GPS に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

WhiteBalance(イメージキットコントロール／Exif プロパティ)

【機能】
撮影時に設定されたホワイトバランスモードを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**Exif**->**WhiteBalance** [= *int*]
(2)Delphi　*imagekitcontrolname*.**FileIO.Exif.WhiteBalance** [= *Integer*]

【設定値】

値	説明
0	ホワイトバランス自動
1	ホワイトバランス手動

【解説】
FileIO.GetImageFileType(Mem)メソッドを実行することにより、プロパティに値が設定されます。
Exif の撮影条件に関するタグです。

【値の設定】 実行時

【値の参照】 実行時

PDF（イメージキットコントロール／カスタム階層プロパティ）

【機能】
PDF 形式のファイルを保存する際に使用します。

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Application	アプリケーション名
Author	作成者名
DocumentHeight	文書の高さのサイズ
DocumentSize	文書のサイズ
DocumentWidth	文書の幅のサイズ
EnableCopy	文書内容のコピー・抽出を許可するかどうかの設定
EnableEdit	文書の編集を許可するかどうかの設定
EnableEditAll	文書の全編集を許可するかどうかの設定
EnablePrint	文書の印刷を許可するかどうかの設定
Keywords	キーワード
Landscape	文書の向き
OwnerPassword	オーナーパスワード
SavePDFFileName	保存するファイル名
Subject	サブタイトル
Title	タイトル
UserPassword	ユーザーパスワード

●メソッド一覧（アルファベット順）

カスタムメソッド	内容
AddImage	PDF の作成で画像を追加
AddPage	PDF の作成でページを追加
Finish	PDF の作成処理を終了
Start	PDF の作成処理を開始

【ImageKit10 ActiveX との違い】
変更されたメソッド：
End --> Finish

Application(イメージキットコントロール／PDF プロパティ)

【機能】
アプリケーション名を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**Application** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PDF.Application** [= *string*]

【設定値】
アプリケーション名。

【解説】
デフォルトは空文字列です。当プロパティは **Start** メソッド実行時に参照されるため、**Start** メソッド実行前に適切な値を設定してください。
当プロパティで設定した文字列は PDF の文書のプロパティの概要の「アプリケーション」に反映されます。

【値の設定】 実行時

【値の参照】 実行時

Author(イメージキットコントロール／PDF プロパティ)

【機能】
作成者名を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->PDF->Author** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PDF.Author** [= *string*]

【設定値】
作成者名。

【解説】
デフォルトは空文字列です。当プロパティは **Start** メソッド実行時に参照されるため、**Start** メソッド実行前に適切な値を設定してください。
当プロパティで設定した文字列は PDF の文書のプロパティの概要の「作成者」に反映されます。

【値の設定】 実行時

【値の参照】 実行時

DocumentHeight(イメージキットコントロール／PDF プロパティ)

【機能】
文書の高さのサイズ取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**DocumentHeight** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PDF.DocumentHeight** [= *Smallint*]

【設定値】
文書の高さのサイズ(mm 単位)。

【解説】
デフォルトは 297 です。297 は A4 サイズの縦向きの高さを表します。
文書のサイズは当プロパティと **DocumentWidth** プロパティ、または **DocumentSize** プロパティにより決まります。当プロパティの設定値を有効にする場合は **DocumentSize** プロパティに空文字列や NULL(C++Builder), nil(Delphi)を設定してください。
当プロパティは **AddPage** メソッド実行時に参照されるため、**AddPage** メソッド実行前に適切な値を設定してください。

【値の設定】 実行時

【値の参照】 実行時

DocumentSize(イメージキットコントロール／PDF プロパティ)

【機能】
文書のサイズを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**DocumentSize** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**PDF**.**DocumentSize** [= *string*]

【設定値】
文書のサイズ。

【解説】
デフォルトは空文字列です。文書のサイズは当プロパティまたは **DocumentWidth** と **DocumentHeight** プロパティの 2 つにより決まります。当プロパティに A4 などのサイズを設定すれば、そちらが有効になり、空文字列や NULL(C++Builder), nil(Delphi)を設定すれば、**DocumentWidth** と **DocumentHeight** プロパティに設定した値が有効になります。
当プロパティは **AddPage** メソッド実行時に参照されるため、**AddPage** メソッド実行前に適切な値を設定してください。

当プロパティに設定可能なサイズは次の通りです。
A0、A1、A2、A3、A4、A5、A6、A7、A8、A9、B0、B1、B2、B3、B4、B5、B6、B7、B8、B9、LETTER、LEGAL
※大文字、小文字は問いません。

【値の設定】 実行時

【値の参照】 実行時

DocumentWidth(イメージキットコントロール／PDF プロパティ)

【機能】
文書の幅のサイズ取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->PDF->DocumentWidth** [= *short*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PDF.DocumentWidth** [= *Smallint*]

【設定値】
文書の幅のサイズ(mm 単位)。

【解説】
デフォルトは 210 です。210 は A4 サイズの縦向きの幅を表します。
文書のサイズは当プロパティと **DocumentHeight** プロパティ、または **DocumentSize** プロパティにより決まります。当プロパティの設定値を有効にする場合は **DocumentSize** プロパティに空文字列や NULL(C++Builder), nil(Delphi)を設定してください。
当プロパティは **AddPage** メソッド実行時に参照されるため、**AddPage** メソッド実行前に適切な値を設定してください。

【値の設定】 実行時

【値の参照】 実行時

EnableCopy(イメージキットコントロール／PDF プロパティ)

【機能】
文書内容のコピー・抽出を許可するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**EnableCopy** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PDF.EnableCopy** [= *Boolean*]

【設定値】

値	説明
True	許可する
False	許可しない

【解説】
デフォルトは True です。当プロパティは **Start** メソッド実行時に参照されるため、**Start** メソッド実行前に適切な値を設定してください。当プロパティの設定値が有効となるのは **OwnerPassword** プロパティに文字列を設定した場合です。**OwnerPassword** プロパティに空文字列を設定した場合は無効です。
当プロパティで設定した値は PDF の文書のプロパティのセキュリティの「内容のコピー」「アクセシビリティのための内容の抽出」に反映されます。

【値の設定】 実行時

【値の参照】 実行時

EnableEdit (イメージキットコントロール／PDF プロパティ)

【機能】
文書の編集を許可するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**EnableEdit** [= *Bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PDF.EnableEdit** [= *Boolean*]

【設定値】

値	説明
True	許可する
False	許可しない

【解説】
デフォルトは True です。当プロパティは **Start** メソッド実行時に参照されるため、**Start** メソッド実行前に適切な値を設定してください。当プロパティの設定値が有効となるのは **OwnerPassword** プロパティに文字列を設定した場合です。**OwnerPassword** プロパティに空文字列を設定した場合は無効です。
当プロパティで設定した値は PDF の文書のプロパティのセキュリティの「フォームフィールドの入力」「署名」「テンプレートページの作成」に反映されます。

【値の設定】 実行時

【値の参照】 実行時

EnableEditAll(イメージキットコントロール／PDF プロパティ)

【機能】
文書の全編集を許可するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->FileIO->PDF->EnableEditAll [= *bool*]
(2)Delphi　*imagekitcontrolname*.FileIO.PDF.EnableEditAll [= *Boolean*]

【設定値】

値	説明
True	許可する
False	許可しない

【解説】
デフォルトは True です。当プロパティは Start メソッド実行時に参照されるため、Start メソッド実行前に適切な値を設定してください。当プロパティの設定値が有効となるのは OwnerPassword プロパティに文字列を設定した場合です。OwnerPassword プロパティに空文字列を設定した場合は無効です。
当プロパティで設定した値は PDF の文書のプロパティのセキュリティの「注釈」「フォームフィールドの入力」「署名」「テンプレートページの作成」に反映されます。

【値の設定】 実行時

【値の参照】 実行時

EnablePrint(イメージキットコントロール／PDF プロパティ)

【機能】
文書の印刷を許可するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**EnablePrint** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**PDF**.**EnablePrint** [= *Boolean*]

【設定値】

値	説明
True	許可する
False	許可しない

【解説】
デフォルトは True です。当プロパティは **Start** メソッド実行時に参照されるため、**Start** メソッド実行前に適切な値を設定してください。当プロパティの設定値が有効となるのは **OwnerPassword** プロパティに文字列を設定した場合です。**OwnerPassword** プロパティに空文字列を設定した場合は無効です。
当プロパティで設定した値は PDF の文書のプロパティのセキュリティの「印刷」に反映されます。

【値の設定】 実行時

【値の参照】 実行時

Keywords(イメージキットコントロール／PDF プロパティ)

【機能】
キーワードを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO->PDF->Keywords** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PDF.Keywords** [= *string*]

【設定値】
キーワード。

【解説】
デフォルトは空文字列です。当プロパティは **Start** メソッド実行時に参照されるため、**Start** メソッド実行前に適切な値を設定してください。
当プロパティで設定した文字列は PDF の文書のプロパティの概要の「キーワード」に反映されます。

【値の設定】 実行時

【値の参照】 実行時

Landscape(イメージキットコントロール／PDF プロパティ)

【機能】
文書の向きを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**Landscape** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**PDF**.**Landscape** [= *Boolean*]

【設定値】

値	説明
True	ランドスケープ(横向き)
False	ポートレート(縦向き)

【解説】
デフォルトは False です。各ページごとに文書の向きの設定を変更することができます。ページによって文書の向きを変更する場合は **AddImage** メソッド実行前に当プロパティに適切な値を設定してください。

【値の設定】 実行時

【値の参照】 実行時

OwnerPassword(イメージキットコントロール／PDF プロパティ)

【機能】
オーナーパスワードを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->FileIO->PDF->OwnerPassword [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.PDF.OwnerPassword [= *string*]

【設定値】
オーナーパスワード。

【解説】
デフォルトは空文字列です。当プロパティは Start メソッド実行時に参照されるため、Start メソッド実行前に適切な値を設定してください。
当プロパティは文書の編集や印刷などを制限するパスワードとなります。
オーナーパスワードとユーザーパスワードを設定する場合、当プロパティには UserPassword プロパティとは異なる値を設定してください。

【値の設定】 実行時

【値の参照】 実行時

SavePDFFileName（イメージキットコントロール／PDF プロパティ）

【機能】
保存する PDF のファイル名を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**SavePDFFileName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PDF.SavePDFFileName** [= *string*]

【設定値】
保存する PDF のファイル名。

【解説】
当プロパティは **Finish** メソッド実行時に参照され、**Finish** メソッドが成功すると当プロパティに設定したファイルが作成されます。

【値の設定】 実行時

【値の参照】 実行時

Subject(イメージキットコントロール／PDF プロパティ)

【機能】
サブタイトルを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**Subject** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO**.**PDF**.**Subject** [= *string*]

【設定値】
サブタイトル。

【解説】
デフォルトは空文字列です。当プロパティは **Start** メソッド実行時に参照されるため、**Start** メソッド実行前に適切な値を設定してください。
当プロパティで設定した文字列は PDF の文書のプロパティの概要の「サブタイトル」に反映されます。

【値の設定】 実行時

【値の参照】 実行時

Title(イメージキットコントロール／PDF プロパティ)

【機能】
タイトルを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**FileIO**->**PDF**->**Title** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**FileIO.PDF.Title** [= *string*]

【設定値】
タイトル。

【解説】
デフォルトは空文字列です。当プロパティは **Start** メソッド実行時に参照されるため、**Start** メソッド実行前に適切な値を設定してください。
当プロパティで設定した文字列は PDF の文書のプロパティの概要の「タイトル」に反映されます。

【値の設定】 実行時

【値の参照】 実行時

UserPassword(イメージキットコントロール／PDF プロパティ)

【機能】
ユーザーパスワードを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->FileIO->PDF->UserPassword [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.FileIO.PDF.UserPassword [= *string*]

【設定値】
ユーザーパスワード。

【解説】
デフォルトは空文字列です。当プロパティは Start メソッド実行時に参照されるため、Start メソッド実行前に適切な値を設定してください。
当プロパティは文書を開く際のパスワードとなります。
ユーザーパスワードのみの設定は不可です。オーナーパスワードまたはオーナーパスワードとユーザーパスワードの 2 つを設定してください。当プロパティには OwnerPassword プロパティとは異なる値を設定してください。

【値の設定】 実行時

【値の参照】 実行時

AddImage（イメージキットコントロール／PDF メソッド）

【機能】
PDF の作成で画像を追加します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**FileIO**->**PDF**->**AddImage**(NativeUInt ImageHandle, long x, long y, long size)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**FileIO.PDF.AddImage**(ImageHandle: THandle; x, y, size: Longint)

【引数】

名称	内容
ImageHandle	対象となる画像の Raw データのハンドル（JPEG と PNG 形式が対象）
x	画像を追加する横方向の位置（左下を原点に mm 単位で指定）
y	画像を追加する縦方向の位置（左下を原点に mm 単位で指定）
size	画像を追加するサイズの指定（mm 単位）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
追加可能な画像は JPEG と PNG です。画像データの縦サイズと横サイズのうち大きいサイズを size に合わせて指定した x,y の位置に画像を設定します。画像データの縦横比は保持したまま、x,y,size は mm 単位での指定です。
当メソッド実行後に画像や位置・サイズを変更するなどして当メソッドを繰り返せば、一つのページ内に複数の画像を追加することも可能です。当メソッドは **Start**、**AddPage**、**Finish** メソッドと共に使用します。

AddPage(イメージキットコントロール／PDF メソッド)

【機能】
PDF の作成でページを追加します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**PDF**->**AddPage**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**PDF**.**AddPage**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
新しいページを追加(作成)します。当メソッドで作成したページに **AddImage** メソッドで画像を追加(埋め込み)します。
当メソッドと **AddImage** メソッドを繰り返せば、複数ページの PDF ファイルを作成することが可能です。
(異なる文書サイズのページの混在も可能です。)
当メソッドは **Start**、**AddImage**、**Finish** メソッドと共に使用します。

Finish(イメージキットコントロール／PDF メソッド)

【機能】
PDF の作成処理を終了します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**PDF**->**Finish**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**PDF**.**Finish**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
SavePDFFileName プロパティに設定したファイル名で保存する終了処理です。当メソッドは **Start**、**AddImage**、**AddPage** メソッドと共に使用します。

【ImageKit10 ActiveX との違い】
メソッドの名称が End から変更されました。

Start(イメージキットコントロール／PDF メソッド)

【機能】
PDF の作成処理を開始します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**FileIO**->**PDF**->**Start**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**FileIO**.**PDF**.**Start**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
当メソッド実行前に必要に応じて **OwnerPassword**、**UserPassword** などのプロパティに適切な値を設定してください。当メソッドは **AddImage**、**AddPage**、**Finish** メソッドと共に使用します。

コード例:
(1)C++Builder
```
VImageKit1->FileIO->FileName = "C:¥¥PNG¥¥load.png";
if (VImageKit1->FileIO->FileLoadAsRawData() == false) return;

VImageKit1->FileIO->PDF->OwnerPassword = "abcd";
VImageKit1->FileIO->PDF->Application = "PDF 作成ツール";
VImageKit1->FileIO->PDF->Author = "Newtone Corp.";
VImageKit1->FileIO->PDF->EnablePrint = true;
if (VImageKit1->FileIO->PDF->Start() == false)
{
  VImageKit1->FreeMemory(VImageKit1->FileIO->ImageHandleRawData);
  return;
}

VImageKit1->FileIO->PDF->DocumentSize = "A4";
VImageKit1->FileIO->PDF->AddPage();
VImageKit1->FileIO->PDF->AddImage(VImageKit1->FileIO->ImageHandleRawData, 30, 50, 100);

VImageKit1->FileIO->PDF->SavePDFFileName = "C:¥¥PDF¥¥save.pdf";
VImageKit1->FileIO->PDF->Finish();

VImageKit1->FreeMemory(VImageKit1->FileIO->ImageHandleRawData);
```

(2)Delphi
```
VImageKit1.FileIO.FileName := 'C:¥PNG¥load.png';
if (VImageKit1.FileIO.FileLoadAsRawData = False) then Exit;

VImageKit1.FileIO.PDF.OwnerPassword := 'abcd';
VImageKit1.FileIO.PDF.Application := 'PDF 作成ツール';
VImageKit1.FileIO.PDF.Author := 'Newtone Corp.';
VImageKit1.FileIO.PDF.EnablePrint := True;
if (VImageKit1.FileIO.PDF.Start = False) then
begin
  VImageKit1.FreeMemory(VImageKit1.FileIO.ImageHandleRawData);
  Exit;
end;
```

```
VImageKit1.FileIO.PDF.DocumentSize := 'A4';
VImageKit1.FileIO.PDF.AddPage;
VImageKit1.FileIO.PDF.AddImage(VImageKit1.FileIO.ImageHandleRawData, 30, 50, 100);

VImageKit1.FileIO.PDF.SavePDFFileName := 'C:¥PDF¥save.pdf';
VImageKit1.FileIO.PDF.Finish;

VImageKit1.FreeMemory(VImageKit1.FileIO.ImageHandleRawData);
```

Layer(イメージキットコントロール／カスタム階層プロパティ)

【機能】
100 階層分のイメージを管理します。階層イメージを利用する場合は事前に基本イメージを設定しておいてください。

「階層イメージの規則」
・基本イメージを表示する際にその上部に表示されます。
・サイズが基本イメージよりも大きい場合は、基本イメージのサイズに合わせられます

●プロパティ一覧(アルファベット順)

カスタムプロパティ	内容
BitCount	イメージの 1 ピクセルあたりのビット数
DXFBlack	DXF イメージの白を黒に置き換えて描画するかどうかの設定
Gray	グレースケールの有無
Height	イメージの縦のピクセル数
ImageHandle	イメージのメモリハンドル
ImageKind	イメージの種類
ImageSize	イメージサイズのバイト数
Mask1632	イメージのカラーマスクの種類
PalCount	イメージの使用パレット数
Picture	TPicture と ImageKit 間でラスタイメージやベクトルイメージのやり取りを行う
ShowInDisp	イメージキットコントロールへの表示設定
ShowInMagnifier	虫眼鏡ウィンドウへの表示設定
ShowInPanWindow	パンウィンドウへの表示設定
StartX	イメージの横方向の表示開始位置
StartY	イメージの縦方向の表示開始位置
TransBlue	ラスタイメージの透過色の青
TransGreen	ラスタイメージの透過色の緑
Transparent	ラスタイメージの透過の設定
TransRed	ラスタイメージの透過色の赤
Width	イメージの横のピクセル数
WidthByte	イメージの 1 ラインのバイト数
Xdpi	イメージの横方向の 1 インチあたりのピクセル数
Ydpi	イメージの縦方向の 1 インチあたりのピクセル数

BitCount(イメージキットコントロール／Layer プロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの 1 ピクセルあたりのビット数を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->Layer[Index]->BitCount [= *short*]
(2)Delphi　*imagekitcontrolname*.Layer[Index].BitCount [= *Smallint*]

※Index は 0～99

【参照値】
イメージの 1 ピクセルあたりのビット数。

値	説明
1	2 値
4	16 色
8	256 色
16	16 ビットカラー
24	24 ビットカラー
32	32 ビットカラー

【解説】
ImageHandle プロパティに値が設定されると、BitCount プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

DXFBlack(イメージキットコントロール／Layer プロパティ)

【機能】
DXF イメージを描画する際に白を黒に置き換えて描画するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]->DXFBlack** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].DXFBlack** [= *Boolean*]

※Index は 0～99

【設定値】

値	説明
True	置き換えを有効
False	置き換えを無効

【解説】
デフォルトは True です。
True の場合は、DXF イメージの白を黒に置き換えて描画します。

【値の設定】 実行時

【値の参照】 実行時

Gray(イメージキットコントロール／Layer プロパティ)

【機能】
ImageHandle プロパティに設定されたイメージがグレースケールであるかどうかを取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]->Gray** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].Gray** [= *Boolean*]

※Index は 0～99

【参照値】

値	説明
True	グレースケール
False	グレースケールでない

【解説】
ImageHandle プロパティに値が設定されると、Gray プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

Height,Width(イメージキットコントロール／Layer プロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの縦横のサイズをピクセル単位で取得します。

【書式】
※Height にて説明(Width も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]->Height** [*= int*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].Height** [*= Integer*]

※Index は 0～99

【参照値】
Height は縦方向のピクセル数。
Width は横方向のピクセル数。

【解説】
ImageHandle プロパティに値が設定されると、Height,Width プロパティは自動的に更新されます。

【値の設定】 不可

【値の参照】 実行時

ImageHandle(イメージキットコントロール／Layer プロパティ)

【機能】
イメージのメモリハンドルを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]**->**ImageHandle** [= *NativeUInt*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].ImageHandle** [= *THandle*]

※Index は 0～99

【設定値】
イメージのメモリハンドル。

【解説】
LayerNo プロパティに 0～99 を設定してファイルをロードすると **ImageHandle** プロパティに値が設定されます。ただし、基本イメージが事前に設定されている場合が対象です(**LayerNo** プロパティに-1 を設定してファイルをロードするなど)。
ImageHandle プロパティの指すイメージを無効にする場合は 0 を設定してください。そうすることにより自動的にメモリが解放されます。また、**ImageHandle** プロパティについては **FreeMemory** メソッドを使用して明示的にメモリを解放しなくても、再設定時やコントロールが破棄される時に自動的に解放されます。

例(Delphi):
```
//VImageKit1.ImageHandle が 0 ではない(何らかのイメージが設定されている)
VImageKit1.LayerNo := 0;
VImageKit1.Layer[0].Transparent := False;
VImageKit1.FileIO.LoadFile(vikLoad);
//VImageKit1.Layer[0].ImageHandle にロードしたメモリハンドルが設定される
```

【値の設定】 実行時

【値の参照】 実行時

ImageKind(イメージキットコントロール／Layer プロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの種類を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]->ImageKind** [= *TVIkImageKind*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].ImageKind** [= *TVIkImageKind*]

※Index は 0～99

【TVIkImageKind 型】
ユニット
IkInit
type
TVIkImageKind = (vikInvalidImage, vikRasterImage, vikVectorWMF, vikVectorEMF, vikVectorDXF, vikVectorSVG, vikVectorSXF);

【参照値】

値	説明
vikInvalidImage	不明
vikRasterImage	ラスタイメージ
vikVectorWMF	WMF
vikVectorEMF	EMF
vikVectorDXF	DXF
vikVectorSVG	SVG
vikVectorSXF	SXF

【解説】
ImageHandle プロパティに値が設定されると、**ImageKind** プロパティは自動的に更新されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikErrorImage, ikRasterImage, ikVectorWMF, ikVectorEMF, ikVectorDXF, ikVectorSVG, ikVectorSXF)。
・ikErrorImage が vikInvalidImage に変更されました。

ImageSize（イメージキットコントロール／Layer プロパティ）

【機能】
ImageHandle プロパティに設定されたイメージのサイズをバイト単位で取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]->ImageSize** [= *int*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].ImageSize** [= *Integer*]

※Index は 0～99

【参照値】
イメージサイズのバイト数。

【解説】
ImageHandle プロパティに値が設定されると、ImageSize プロパティは自動的に更新されます。

【値の設定】 不可

【値の参照】 実行時

Mask1632(イメージキットコントロール／Layer プロパティ)

【機能】
ImageHandle プロパティに設定された 16 ビットカラーおよび 32 ビットカラーのイメージのカラーマスクの種類を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]->Mask1632** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].Mask1632** [= *Smallint*]

※Index は 0～99

【参照値】
16,32 ビットカラーの時のカラーマスクの種類。

値	説明
0	BitCount プロパティが 16,32 でない
1	カラーマスクあり(RGB555 フォーマット)
2	カラーマスクあり(RGB565 フォーマット)
3	カラーマスクなし(RGB555 フォーマット)
4	カラーマスクあり(RGB888 フォーマット)
5	カラーマスクなし(RGB888 フォーマット)

【解説】
ImageHandle プロパティに値が設定されると、Mask1632 プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

PalCount(イメージキットコントロール／Layer プロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの使用パレット数を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->Layer[Index]->PalCount [= *short*]
(2)Delphi　*imagekitcontrolname*.Layer[Index].PalCount [= *Smallint*]

※Index は 0～99

【参照値】
イメージの使用パレット数。
BitCount プロパティが 8 以下の場合が対象となり、それ以外は 0 となります。

【解説】
ImageHandle プロパティに値が設定されると、PalCount プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

Picture(イメージキットコントロール／Layer プロパティ)

【機能】
TPicture と ImageKit 間でラスタイメージやベクトルイメージのやり取りをするために使用します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]->Picture** [= *TPicture**]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].Picture** [= *TPicture*]

※Index は 0～99

【設定値】
TPicture。
TPicture については Delphi や C++Builder のヘルプを参照してください。

【解説】
ラスタイメージをやり取りするコードを示します。

C++Builder の例:
(1)TImage → TVImageKit
```
VImageKit1->Layer[0]->Picture = Image1->Picture;
```
(2)TVImageKit → TImage
```
Image1->Picture = VimageKit1->Layer[0]->Picture;
```

Delphi の例:
(1)TImage → TVImageKit
```
VImageKit1.Layer[0].Picture := Image1.Picture;
```
(2)TVImageKit → TImage
```
Image1.Picture := VImageKit1.Layer[0].Picture;
```

【値の設定】 実行時

【値の参照】 実行時

ShowInDisp,ShowInMagnifier,ShowInPanWindow(イメージキットコントロール／Layer プロパティ)

【機能】
イメージを表示するかどうかを取得または設定します。

【書式】
※**ShowInDisp** にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]**->**ShowInDisp** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].ShowInDisp** [= *Boolean*]

※Index は 0～99

【設定値】
ShowInDisp はイメージをイメージキットコントロールに表示するかどうかを設定します。
ShowInMagnifier はイメージを虫眼鏡ウィンドウに表示するかどうかを設定します。
ShowInPanWindow はイメージをパンウィンドウに表示するかどうかを設定します。

値	説明
True	イメージを表示する
False	イメージを表示しない

【解説】
デフォルトは True です。
対象となるイメージは **Layer[Index].ImageHandle** プロパティが指すイメージのことです。

【値の設定】 実行時

【値の参照】 実行時

StartX,StartY(イメージキットコントロール／Layer プロパティ)

【機能】
イメージの縦・横の開始位置(ピクセル)を取得または設定します。

【書式】
※**StartX** にて説明(**StartY** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]**->**StartX** [= *int*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].StartX** [= *Integer*]

※Index は 0～99

【設定値】
イメージの縦横の表示開始位置(基本イメージの左上からのピクセル値)。
デフォルトは 0。

【解説】
基本イメージが設定されていない場合は無効です。
対象となるイメージは **Layer[Index].ImageHandle** プロパティが指すイメージのことです。

使用例(Delphi)

```
VImageKit1.Layer[0].StartX := 300;
VImageKit1.Layer[0].StartY := 200;
VImageKit1.Refresh;
```

【値の設定】 実行時

【値の参照】 実行時

TransBlue,TransGreen,TransRed(イメージキットコントロール／Layer プロパティ)

【機能】
ラスタイメージの透過色を取得または設定します。

【書式】
※**TransBlue** にて説明(**TransGreen**、**TransRed** も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->**Layer[Index]**->**TransBlue** [= *short*]
(2)Delphi　　*imagekitcontrolname*.**Layer[Index].TransBlue** [= *Smallint*]

※Index は 0～99

【設定値】
TransBlue は透過色の青。
TransGreen は透過色の緑。
TransRed は透過色の赤。

0～255。

【解説】
複数のラスタイメージを重ね合わせする場合に使用します。設定した透過色の部分に別の階層のイメージが表示されます。
透過対象イメージは 8 ビットカラーもしくは 24 ビットカラーでなければなりません。
透過対象イメージは **Layer[Index].ImageHandle** プロパティが指すラスタイメージのことです(ベクトルイメージが設定されている場合は無効です)。
透過色を設定する場合は、透過対象イメージよりも先に透過色を設定する必要があります。

透過色を白にする例(Delphi):

```
VImageKit1.Layer[0].TransBlue := 255;
VImageKit1.Layer[0].TransGreen := 255;
VImageKit1.Layer[0].TransRed := 255;
VImageKit1.Layer[0].Transparent := True;
VImageKit1.LayerNo := 0;
VImageKit1.FileIO.LoadFile(vikLoad);
```

【値の設定】 実行時

【値の参照】 実行時

Transparent（イメージキットコントロール／Layer プロパティ）

【機能】
ラスタイメージの透過を有効にするかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Layer[Index]->Transparent** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Layer[Index].Transparent** [= *Boolean*]

※Index は 0～99

【設定値】

値	説明
True	透過を有効
False	透過を無効

【解説】
デフォルトは True です。
True の場合、**TransBlue**,**TransGreen**,**TransRed** プロパティで設定した色を透過色とすることができます。
透過色を無効にして単純にラスタイメージを重ねる場合は False を設定してください。
透過対象イメージの **Layer[Index].ImageHandle** プロパティがベクトルイメージの場合は無効です。

【値の設定】 実行時

【値の参照】 実行時

WidthByte(イメージキットコントロール／Layer プロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの 1 ラインのバイト数を取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->Layer[Index]->WidthByte [= *int*]
(2)Delphi　*imagekitcontrolname*.Layer[Index].WidthByte [= *Integer*]

※Index は 0～99

【参照値】
イメージの 1 ラインのバイト数。

【解説】
ImageHandle プロパティに値が設定されると、WidthByte プロパティは自動的に更新されます。
ベクトルイメージの場合は無効です。

【値の設定】 不可

【値の参照】 実行時

Xdpi,Ydpi(イメージキットコントロール／Layer プロパティ)

【機能】
ImageHandle プロパティに設定されたイメージの縦横の 1 インチあたりのピクセル数を取得または設定します。

【書式】
※Xdpi にて説明(Ydpi も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->Layer[Index]->Xdpi [= *int*]
(2)Delphi　*imagekitcontrolname*.Layer[Index].Xdpi [= *Integer*]

※Index は 0～99

【参照値】
Xdpi は横方向の 1 インチあたりのピクセル数。
Ydpi は縦方向の 1 インチあたりのピクセル数。

【解説】
ImageHandle プロパティに値が設定されると、Xdpi,Ydpi プロパティは自動的に更新されます。
また、イメージの解像度を変更する場合は LayerNo プロパティにインデックス番号を設定してから SetDpi メソッドを実行します。

【値の設定】 実行時

【値の参照】 実行時

Magnifier（イメージキットコントロール／カスタム階層プロパティ）

【機能】
虫眼鏡処理を実現します。

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Edge	虫眼鏡ウィンドウの縁の描画設定
Height	虫眼鏡ウィンドウの縦方向のサイズ
ShowCursor	マウスカーソルの表示設定
Style	虫眼鏡ウィンドウの形状
Width	虫眼鏡ウィンドウの横方向のサイズ

【ImageKit7/8/9/10 ActiveX との違い】
変更されたプロパティ： Type --> Style

●メソッド一覧（アルファベット順）

カスタムメソッド	内容
Show	虫眼鏡ウィンドウの表示

Edge(イメージキットコントロール／Magnifier プロパティ)

【機能】
虫眼鏡ウィンドウの縁を描画するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Magnifier**->**Edge** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Magnifier.Edge** [= *Boolean*]

【設定値】

値	説明
True	縁を描画する
False	縁を描画しない

【解説】
デフォルトは False です。
True の場合、縁の色は黒固定です。

【値の設定】 実行時

【値の参照】 実行時

Height,Width(イメージキットコントロール／Magnifier プロパティ)

【機能】
虫眼鏡ウィンドウの縦方向と横方向のサイズを取得または設定します。

【書式】
※**Height** にて説明(**Width** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Magnifier**->**Height** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Magnifier**.**Height** [= *Smallint*]

【設定値】
Height は縦方向のサイズ(ピクセル)
Width は横方向のサイズ(ピクセル)

【解説】
デフォルトは両方とも 100 です。

【値の設定】 実行時

【値の参照】 実行時

ShowCursor(イメージキットコントロール／Magnifier プロパティ)

【機能】
マウスカーソルを表示するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Magnifier**->**ShowCursor** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Magnifier.ShowCursor** [= *Boolean*]

【設定値】

値	説明
True	マウスカーソルを表示する
False	マウスカーソルを表示しない

【解説】
デフォルトは False です。
False の場合、イメージキットコントロール内にマウスカーソルは表示されません。

【値の設定】 実行時

【値の参照】 実行時

Style(イメージキットコントロール／Magnifier プロパティ)

【機能】
虫眼鏡ウィンドウの形状を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Magnifier**->**Style** [= *TVIkMagnifierStyle*]
(2)Delphi　*imagekitcontrolname*.**Magnifier.Style** [= *TVIkMagnifierStyle*]

【TVIkMagnifierStyle 型】
ユニット
IkInit
type
TVIkMagnifierStyle = (vikRectangleMagnifier, vikRoundRectMagnifier, vikEllipseMagnifier);

【設定値】

値	説明
vikRectangleMagnifier	四角形
vikRoundRectMagnifier	角丸四角形
vikEllipseMagnifier	楕円

【解説】
デフォルトは四角形です。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
・プロパティの名称が Type から変更されました。
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikRectangleMagnifier, ikRoundRectMagnifier, ikEllipseMagnifier)。

Show(イメージキットコントロール／Magnifier メソッド)

【機能】
Height,Width プロパティで設定したサイズで虫眼鏡ウィンドウを表示します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Magnifier**->**Show**(bool Show, int X, int Y, double Scale)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Magnifier**.**Show**(Show: Boolean; X, Y: Integer; Scale: Double);

【引数】

名称	内容
Show	虫眼鏡ウィンドウの表示設定 [True:虫眼鏡ウィンドウを表示、False:虫眼鏡ウィンドウを消去]
X,Y	虫眼鏡ウィンドウの中心位置(ピクセル)
Scale	虫眼鏡ウィンドウに表示するイメージの比率(実寸は 1.0)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Height,Width プロパティに適切な値が設定されている必要があります。
Show=False の場合、X,Y,Scale の値は無視されます。
ShowInMagnifier プロパティが True に設定されているイメージを表示します(基本イメージ、**Layer** イメージが対象)。
イメージを表示する順番は、基本イメージ、**Layer** イメージ(インデックスが小さい方から大きい方へ)となります。
ただし、基本イメージが設定されていない場合、**Layer** イメージは表示されません。

虫眼鏡ウィンドウを表示するコード例:
(1)C++Builder

```
void _fastcall TForm1::VImageKit1MouseMoveImage(TObject *Sender, TShiftState Shift, int OriginX, int OriginY, int
ALeft, int ATop, int ARight, int ABottom, double ScaleWidth, double ScaleHeight, int X, int Y)
{
  //虫眼鏡
  VImageKit1->Magnifier->Width = 100;
  VImageKit1->Magnifier->Height = 100;
  VImageKit1->Magnifier->Show(true, X, Y, 2);
}
```

(2)Delphi

```
procedure TForm1.VImageKit1MouseMoveImage(Sender: TObject; Shift: TShiftState; OriginX, OriginY, ALeft, ATop,
ARight, ABottom: Integer; ScaleWidth, ScaleHeight: Double; X, Y: Integer);
begin
  //虫眼鏡
  VImageKit1.Magnifier.Width := 100;
  VImageKit1.Magnifier.Height := 100;
  VImageKit1.Magnifier.Show(True, X, Y, 2);
end;
```

PanWindow（イメージキットコントロール／カスタム階層プロパティ）

【機能】
パンウィンドウ処理を実現します。

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Caption	パンウィンドウのキャプションの設定
CloseBox	パンウィンドウの閉じるボタンの設定
Color	パンウィンドウ内の無効領域の色（ベクトルイメージ描画時に使用）
Cursor	パンウィンドウの描画領域内におけるマウスカーソルの形状
EnableClick	パンウィンドウでマウスクリックを有効にするかどうかの設定
Enabled	マウスやキーボードイベントへの応答可否
Handle	パンウィンドウのウィンドウハンドル
Left	パンウィンドウを表示する左位置の設定
RectColor	パンウィンドウ内の矩形を表す線の色
RectCursor	パンウィンドウの矩形領域内におけるマウスカーソルの形状
RectReverse	パンウィンドウの表示時に矩形領域の反転の有無を設定
RefineImage	パンウィンドウ内のイメージを高精彩に表示するかどうかの設定
Size	パンウィンドウのサイズの設定
Top	パンウィンドウを表示する上位置の設定

【ImageKit7/8/9/10 ActiveX との違い】
削除されたプロパティ： MouseCursorFile, RectMouseCursorFile
※MouseCursorFile プロパティは Cursor プロパティで代用
※RectMouseCursorFile プロパティは RectCursor プロパティで代用

変更されたプロパティ:
Hwnd --> Handle
MouseCursorType --> Cursor
RectMouseCursorType --> RectCursor

●メソッド一覧（アルファベット順）

カスタムメソッド	内容
Show	パンウィンドウの表示（非表示）

Caption(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウのキャプションを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PanWindow->Caption** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**PanWindow.Caption** [= *string*]

【設定値】
パンウィンドウのキャプションに表示する文字列。

【解説】
Caption プロパティに空の文字列を設定し **CloseBox** プロパティを False にすると、タイトルバーが表示されなくなり、マウスでパンウィンドウの移動ができなくなります。
Caption プロパティは以前の ImageKit で提供していたディスプレイコントロールの **PanWindowCaption** プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

CloseBox(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウに閉じるボタンを付加するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PanWindow->CloseBox** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**PanWindow.CloseBox** [= *Boolean*]

【設定値】

値	説明
True	閉じるボタン(×ボタン)を付加する
False	閉じるボタン(×ボタン)を付加しない

【解説】
CloseBox プロパティは以前の ImageKit で提供していたディスプレイコントロールの **PanWindowCloseBox** プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

Color,RectColor(イメージキットコントロール／PanWindow プロパティ)

【機能】
色情報を取得または設定します。

【書式】
※Color にて説明(RectColor も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->PanWindow->Color [= *TColor*]
(2)Delphi　*imagekitcontrolname*.PanWindow.Color [= *TColor*]

【設定値】
Color はパンウィンドウ内の無効領域の色(ベクトルイメージ描画時に使用)。
RectColor はパンウィンドウ内の矩形を表す線の色。

【解説】
値を設定する場合は、色定数(clRed など)や RGB(Red,Green,Blue)の戻り値などを与えます。
RectColor は、表示するイメージが白黒 2 値の場合には設定した色がそのまま有効になりますが、カラーの場合は矩形枠の下地の色によっては反転します。また、Color プロパティとの組み合わせにより矩形枠が表示されない場合がありますので、ご注意ください。
TColor 型については Delphi や C++Builder のヘルプを参照してください。
Color,RectColor プロパティは以前の ImageKit で提供していたディスプレイコントロールの PanWindowColor,PanWindowRectColor プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

Cursor,RectCursor(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウのマウスカーソルの形状を取得または設定します。

【書式】
※Cursor にて説明(RectCursor も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->PanWindow->Cursor [= *TCursor*]
(2)Delphi　　*imagekitcontrolname*.PanWindow.Cursor [= *TCursor*]

【設定値】
crDefault や crArrow など。詳しくは Delphi や C++Builder のヘルプを参照のこと。
Cursor プロパティのデフォルトは crArrow、RectCursor プロパティのデフォルトは crSizeAll です。

【解説】
Cursor プロパティはパンウィンドウの描画領域内におけるマウスカーソルの形状を表します。
RectCursor プロパティはパンウィンドウの矩形領域内のマウスカーソルの形状を表します。設定したカーソル形状が有効になるのは、パンウィンドウ内でマウスのボタンが押された場合です。
カスタムカーソルを割り当てる場合は、Delphi や C++Builder のヘルプの TControl.Cursor の例を参照してください。

Cursor プロパティは以前の ImageKit で提供していたディスプレイコントロールの PanWindowCursor プロパティに相当します。
RectCursor プロパティは以前の ImageKit で提供していたディスプレイコントロールの PanWindowRectCursor プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
MouseCursorType, MouseCursorTypeFile プロパティの機能が Cursor プロパティに、RectMouseCursorType, RectMouseCursorTypeFile プロパティの機能が RectCursor プロパティにそれぞれ統一されました。

EnableClick(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウでマウスクリックを有効にするかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PanWindow->EnableClick** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**PanWindow.EnableClick** [= *Boolean*]

【設定値】

値	説明
True	マウスクリックを有効
False	マウスクリックを無効

【解説】
デフォルトは False です。
True の場合、パンウィンドウの矩形領域外でマウスを左クリックすると、クリックされた位置を中心にイメージキットコントロールにイメージを表示します。また、それに伴いパンウィンドウの矩形領域も移動します。
EnableClick プロパティは以前の ImageKit で提供していたディスプレイコントロールの **ClickPanWindow** プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

Enabled(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウのマウスやキーボードイベントに応答するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PanWindow->Enabled** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**PanWindow.Enabled** [= *Boolean*]

【設定値】

値	説明
True	マウスやキーボードイベントを有効
False	マウスやキーボードイベントを無効

【解説】
デフォルトは True です。
True を設定するとマウスダウンやキーダウンなどのイベントに応答できますが、False の場合は該当イベントが発生しません。

【値の設定】 実行時

【値の参照】 実行時

Handle(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウのウィンドウハンドルを取得します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PanWindow->Handle** [= *HWND*]
(2)Delphi　*imagekitcontrolname*.**PanWindow.Handle** [= *HWND*]

【参照値】
パンウィンドウのウィンドウハンドル。

【解説】
Handle プロパティは以前の ImageKit で提供していたディスプレイコントロールの **PanWindowHandle** プロパティに相当します。

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
プロパティの名称が **Hwnd** から変更されました。

Left,Top(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウを表示する左上位置をピクセル単位で取得または設定します。

【書式】
※**Left** にて説明(**Top** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**PanWindow**->**Left** [= *int*]
(2)Delphi　*imagekitcontrolname*.**PanWindow.Left** [= *Integer*]

【設定値】
スクリーン左上からのパンウィンドウの左上位置(ピクセル単位)。
Left は、左位置
Top は、上位置

【解説】
パンウィンドウを表示する位置を指定します。パンウィンドウのサイズは **Size** プロパティで設定します。パンウィンドウを使用すると、イメージキットコントロールに表示しているイメージ全体を簡単に把握することができます。
Left,**Top** プロパティは以前の ImageKit で提供していたディスプレイコントロールの **PanWindowLeft**,**PanWindowTop** プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

RectReverse(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウの矩形領域内を RGB 反転するかどうか取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PanWindow->RectReverse** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**PanWindow.RectReverse** [= *Boolean*]

【設定値】

値	説明
True	パンウィンドウの矩形領域内を RGB 反転する
False	パンウィンドウの矩形領域内を RGB 反転しない

【解説】
デフォルトは False です。True にするとパンウィンドウの矩形領域内を RGB 反転させて表示します。
RectReverse プロパティは以前の ImageKit で提供していたディスプレイコントロールの **PanWindowRectRev** プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

RefineImage(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウ内のイメージを高精彩に表示するかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PanWindow->RefineImage** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**PanWindow.RefineImage** [= *Boolean*]

【設定値】

値	説明
True	高精彩に表示する
False	そのまま表示する

【解説】
デフォルトは True です。
以前の ImageKit で提供していたディスプレイコントロールの **PanWindowRefine1BitImage** プロパティに相当しますが、**RefineImage** プロパティはビット数に関係なく補間処理を行います。
このプロパティは IK7 との互換性保持のために残されていますので、使用しないことをお勧めいたします。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7 ActiveX/VCL との違い】
IK7 と同じ動作にする場合は **StretchMode** プロパティに 0(vikRefineImage)を設定します。

Size(イメージキットコントロール／PanWindow プロパティ)

【機能】
パンウィンドウのサイズをピクセル単位で取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PanWindow**->**Size** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PanWindow.Size** [= *Smallint*]

【設定値】
パンウィンドウのサイズ(ピクセル単位)。

【解説】
イメージキットコントロールに関連づけられているイメージの縦横のうち、大きい方を基準としてスケーリングします。縦長のイメージであれば **Size** プロパティで設定した値は縦方向の大きさになり、横方向の大きさはイメージの縦横比から自動的に計算します。
Size プロパティは以前の ImageKit で提供していたディスプレイコントロールの **PanWindowSize** プロパティに相当します。

【値の設定】 実行時

【値の参照】 実行時

Show(イメージキットコントロール／PanWindow メソッド)

【機能】

Left,**Top** プロパティで設定した位置に、**Size** プロパティで設定したサイズでパンウィンドウを表示します。

【書式】

(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PanWindow**->**Show**(bool Show)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PanWindow**.**Show**(Show: Boolean)

【引数】

名称	内容
Show	パンウィンドウの表示設定 ［ True:パンウィンドウを表示、False:パンウィンドウを消去 ］

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

引数が True の場合には、**Left**,**Top**,**Size** プロパティに適切な値が設定されている必要があります。実寸表示以外の場合は無効です。
ShowInPanWindow プロパティが True に設定されているイメージを表示します(基本イメージ、**Layer** イメージが対象)。
イメージを表示する順番は、基本イメージ、**Layer** イメージ(インデックスが小さい方から大きい方へ)となります。
ただし、基本イメージが設定されていない場合、**Layer** イメージは表示されません。
イメージキットコントロールを配置したフォームをモーダルで表示する場合は、フォームを表示した後でパンウィンドウを表示するようにしてください。

Show メソッドは以前の ImageKit で提供していたディスプレイコントロールの **ShowPanWindow** メソッドに相当します。

PrintDraw（イメージキットコントロール／カスタム階層プロパティ）

【機能】
描画先オブジェクトに対して図形や文字などを描画する機能やプリンタを制御する機能を提供します。

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Alpha1	TextColor1 プロパティのアルファ値
Alpha2	TextColor2 プロパティのアルファ値
BackColor	ペンが実線以外の場合の線と線の間の色、およびブラシがハッチパターンの場合の背景色
BrushColor	塗りつぶすブラシの色
BrushStyle	塗りつぶすブラシの模様
ButtonName	印刷処理中のダイアログボックスに表示するボタンの名称
Caption	印刷処理中のダイアログボックスに表示するキャプションの名称
CharAngle	文字の回転角度
CharExtra	文字の間隔
CharSet	フォントで使用する文字セット
Collate	印刷ダイアログで処理する[部単位で印刷]のチェックボックス
ColorMode	カラープリンタでカラーを使用するかモノクロを使用するかを指定
Copies	印刷部数
CustomPaperHeight	PaperSize プロパティが 256 の時の用紙の縦幅
CustomPaperWidth	PaperSize プロパティが 256 の時の用紙の横幅
DevMode	指定したプリンタの DEVMODE 構造体へのハンドル
DevNames	指定したプリンタの DEVNAMES 構造体へのハンドル
Direction	文字列の描画方向
DocName	印刷するドキュメント名
Duplex	プリンタが両面印刷をサポートしている場合、両面印刷を行うかどうかを指定
FontBold	フォントの太字設定
FontItalic	フォントの斜体設定
FontName	テキストを描画する際のフォント名
FontSize	テキストを描画する際のフォントのサイズ
FontStrikeOut	テキストの打ち消し線設定
FontUnderline	テキストの下線設定
FromPage	印刷ダイアログで処理する開始ページ
Handle	プリンタのデバイスコンテキスト
HCentering	文字列の水平センタリングを設定
HotkeyPrefix	プリフィックス文字をどう扱うかを設定
MaxPage	印刷ダイアログで処理する最大ページ
MinPage	印刷ダイアログで処理する最小ページ
Message	印刷処理中のダイアログボックスに表示するメッセージ名
Options	印刷ダイアログの初期設定
Orientation	印刷の向き
PaperBin	プリンタの用紙トレイ
PaperSize	プリンタの用紙サイズ
PenColor	描画するペンの色を設定
PenMode	描かれるペンの色をキャンバス上の色とどのように対応させるかを設定
PenStyle	描画するペンのスタイルを設定
PenWidth	描画するペンの幅を設定
PortName	プリンタのポート名
Ports	EnumPorts メソッドで取得されるポート名のリスト
PrinterName	プリンタ名
PrinterIndex	EnumPrinters メソッドで取得されるプリンタのデフォルトを示すインデックス
Printers	EnumPrinters メソッドで取得されるプリンタ名のリスト
PrintFileName	プリンタの情報を保存するファイル名
PrintRange	印刷ダイアログの印刷範囲の種類
PrintToFile	印刷ダイアログで処理する[ファイルへ出力]のチェックボックス

RotateString	文字単位で回転するか文字列単位で回転するかを設定
Text	描画文字列を設定
TextColor1	テキストの描画色およびブラシの色
TextColor2	テキストの背景色およびブラシの色
ToPage	印刷ダイアログで処理する終了ページ数
Transparent	テキストやハッチブラシの背景および線と線の間隔を描画する時の透過設定
VCentering	文字列の垂直センタリングを設定
XResolution	プリンタの X 方向の解像度
YResolution	プリンタの Y 方向の解像度
Zoom	スケール時の百分率

描画先オブジェクト:S=スクリーン、P=プリンタ、M=メモリハンドル

【ImageKit7/8/9/10 ActiveX との違い】
変更されたプロパティ: Hdc --> Handle

●メソッド一覧(アルファベット順)

カスタムメソッド	内容
Arc	弧の描画(S,P,M)
Chord	弓形の描画(S,P,M)
ClearProperty	PrintDraw プロパティの初期化
DrawFocusRect	フォーカス付き矩形枠の描画(S)
DrawString	GDI+を使用したテキストの描画(S,P,M)
DrawText	矩形内にテキストを描画(S,P,M)
Ellipse	矩形に内接する円の描画(S,P,M)
EnumPaperBins	プリンタの用紙トレイ名リストを取得
EnumPaperSizes	プリンタの用紙サイズ名リストを取得
EnumPorts	ポートの列挙
EnumPrinters	インストールされているプリンタの列挙
EnumResolutions	プリンタの解像度リストを取得
FillRect	矩形の塗りつぶし(S,P,M)
FrameRect	ブラシによる矩形の描画(S,P,M)
GetArrayNum	列挙する情報に応じて必要な配列の要素数を取得
GetDefaultPrinter	通常使うプリンタの取得
GetDevModeHandle	指定したプリンタの DEVMODE 構造体のハンドルと DEVNAMES 構造体のハンドルを取得
GetDevModeInfo	DEVMODE 構造体のハンドルから印刷情報を取得
GetDevNamesInfo	DEVNAMES 構造体のハンドルからプリンタ名とポート名を取得
GetImageFromHdc	デバイスコンテキスト(hDC)からイメージを取得
GetPaperSize	指定したプリンタの用紙の有効印字領域、高さ、幅を取得
GetPixel	指定したピクセルから RGB 値を取得(S,M)
GetPrinterPort	プリンタのポートの取得
GetTextExtent	指定した文字情報から描画する文字列の高さと幅を取得(S,P)
ImageOut	デバイスコンテキスト(hDC)に対するイメージの描画(S,P)
ImageOutToHwnd	ウィンドウハンドル(hWnd)に対するイメージの描画(S)
Line	直線の描画(S,P,M)
MeasureString	指定した文字情報から描画する文字列の高さと幅を取得(GDI+) (S,P)
Paint	指定した色を別の色に設定(S,P,M)
Pie	扇形の描画(S,P,M)
PolyBezier	ベジェ曲線の描画(S,P,M)
Polygon	多角形の描画(S,P,M)
Polyline	連続線の描画(S,P,M)
PreviewInit	印刷プレビュー時の DPI を設定
PrintAbortDoc	印刷ジョブの終了
PrintCreateDC	デバイスコンテキストの作成
PrintDeleteDC	デバイスコンテキストの削除
PrintDialog	デバイスコンテキストの作成(印刷ダイアログ表示)
PrintEndDoc	プリントの終了
PrintEndPage	ページの終了

PrintStartDoc	プリントの開始
PrintStartPage	ページの開始
Rectangle	矩形の描画(S,P,M)
ReleaseDevModeHandle	DEVMODE 構造体のハンドルと DEVNAMES 構造体のハンドルを解放
RoundRect	角が丸い矩形の描画(S,P,M)
SaveDevModeHandle	DEVMODE 構造体のハンドルと DEVNAMES 構造体のハンドルをファイルに保存
SavePrinterInfo	プリンタの情報を設定ファイルに保存
SetDefaultPrinter	通常使うプリンタを設定
SetDevModeInfo	DEVMODE 構造体のハンドルへ印刷情報を設定
SetPixel	指定したピクセルに RGB 値を設定(S,M)
TextOut	始点を与えてテキストを描画(S,P,M)

Alpha1,Alpha2(イメージキットコントロール／PrintDraw プロパティ)

【機能】
TextColor1,TextColor2 プロパティのアルファ値を設定します。

【書式】
※Alpha1 にて説明(Alpha2 も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->PrintDraw->Alpha1 [= *short*]
(2)Delphi　*imagekitcontrolname*.PrintDraw.Alpha1 [= *Smallint*]

【設定値】
Alpha1 は TextColor1 のアルファ値です(0～255)。
Alpha2 は TextColor2 のアルファ値です(0～255)。

【解説】
255 の場合、完全不透明です。設定した値は DrawString メソッドで有効となります。

【値の設定】 実行時

【値の参照】 不可

BackColor,BrushColor,PenColor,TextColor1,TextColor2
(イメージキットコントロール／PrintDraw プロパティ)

【機能】
ブラシ、ペン、テキストのそれぞれの色情報を設定します。

【書式】
※**BackColor** にて説明(その他も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->**PrintDraw**->**BackColor** [= *TColor*]
(2)Delphi　　*imagekitcontrolname*.**PrintDraw**.**BackColor** [= *TColor*]

【設定値】
BackColor はペンが実線以外の場合の線と線の間の色、およびブラシがハッチパターンの場合の背景色。
BrushColor は塗りつぶすブラシの色。
PenColor は描画するペンの色。

TextColor1:
DrawText,**TextOut** メソッドではテキストの描画色を表します。
DrawString メソッドではブラシの種類により意味合いが異なります。
ソリッドブラシ:ブラシの色
ハッチブラシ:描画される線の色
グラデーションブラシ:線形グラデーションの開始色
※テクスチャブラシでは無効

TextColor2:
DrawText,**TextOut** メソッドではテキストの背景色を表します。
DrawString メソッドではブラシの種類により意味合いが異なります。
ハッチブラシ:線間の領域の色
グラデーションブラシ:線形グラデーションの終了色
※ソリッドブラシとテクスチャブラシでは無効

【解説】
値を設定する場合は、色定数(clRed など)や **RGB**(Red,Green,Blue)の戻り値などを与えます。
デバイスが設定した色を正確に表現できない場合は、近似の色が用いられます。
TColor 型については Delphi や C++Builder のヘルプを参照してください。

【値の設定】 実行時

【値の参照】 不可

BrushStyle(イメージキットコントロール／PrintDraw プロパティ)

【機能】
塗りつぶすブラシの模様を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw->BrushStyle** [= *TVIkBrushStyle*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.BrushStyle** [= *TVIkBrushStyle*]

【TVIkBrushStyle 型】
ユニット
IkInit
type
TVIkBrushStyle = (vikBrushNull, vikBrushSolid, vikBrushHatchBdiagonal, vikBrushHatchCross, vikBrushHatchDiagcross, vikBrushHatchFdiagonal, vikBrushHatchHorizontal, vikBrushHatchVertical);

【設定値】

値	説明
vikBrushNull	塗りつぶしは行わない(BS_NULL)
vikBrushSolid	塗りつぶし(BS_SOLID)
vikBrushHatchBdiagonal	左下から右上へ 45 度の角度で向かう斜線のハッチパターン(BS_HATCHED、HS_BDIAGONAL)
vikBrushHatchCross	縦横の格子線のハッチパターン(BS_HATCHED、HS_CROSS)
vikBrushHatchDiagcross	vikBrushHatchCross のハッチパターンを 45 度回転したもの(BS_HATCHED、HS_DIAGCROSS)
vikBrushHatchFdiagonal	左上から右下へ 45 度の角度で向かう斜線のハッチパターン(BS_HATCHED、HS_FDIAGONAL)
vikBrushHatchHorizontal	横線のハッチパターン(BS_HATCHED、HS_HORIZONTAL)
vikBrushHatchVertical	縦線のハッチパターン(BS_HATCHED、HS_VERTICAL)

( )内の説明は WindowsAPI で使用する定数と同じ意味です。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikBrushNull, ikBrushSolid, ikBrushHatchBdiagonal, ikBrushHatchCross, ikBrushHatchDiagcross, ikBrushHatchFdiagonal, ikBrushHatchHorizontal, ikBrushHatchVertical)。

ButtonName,Caption,DocName,Message(イメージキットコントロール／PrintDraw プロパティ)

【機能】
印刷処理中のダイアログボックスに表示するボタン、キャプション、メッセージおよびドキュメント名を取得または設定します。

【書式】
※**ButtonName** にて説明(**Caption**,**DocName**,**Message** も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->**PrintDraw**->**ButtonName** [= *UnicodeString*]
(2)Delphi　　*imagekitcontrolname*.**PrintDraw.ButtonName** [= *string*]

【設定値】
ButtonName プロパティは、処理中のダイアログボックスのボタンに表示する文字列。
Caption プロパティは、処理中のダイアログボックスのタイトルバーに表示する文字列。
DocName プロパティは、印刷するドキュメント名を表す文字列。
Message プロパティは、処理中のダイアログボックスの中央に表示する文字列。

【解説】
印刷処理中のダイアログボックスを表示した場合はボタンを選択することにより、印刷処理を中止することができます。
ButtonName,**Caption**,**Message** プロパティを設定しなかった場合は、処理中のダイアログボックスは表示されません。
DocName プロパティは印刷するドキュメント名になります。
また、設定したプロパティが有効になるのは **PrintStartDoc** メソッドを実行した場合です。

【値の設定】 実行時

【値の参照】 実行時

CharAngle(イメージキットコントロール／PrintDraw プロパティ)

【機能】
文字の回転角度を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw->CharAngle** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.CharAngle** [= *Smallint*]

【設定値】
DrawTextメソッドの場合
0,90,180,270(反時計回りに回転)
TextOutメソッドの場合
0～360(1 度単位、反時計回りに回転)
DrawStringメソッドの場合
0～360(1 度単位、時計回りに回転)

【解説】
設定した値は **DrawString**,**DrawText**,**TextOut** メソッドで有効となります。

【値の設定】 実行時

【値の参照】 不可

CharExtra(イメージキットコントロール／PrintDraw プロパティ)

【機能】
文字の間隔を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw->CharExtra** [= *int*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.CharExtra** [= *Integer*]

【設定値】
文字間隔(使用しない場合は 0 を設定)。

【解説】
設定した値は **DrawText**,**GetTextExtent**,**TextOut** メソッドで有効となります。
描画対象がスクリーンとメモリハンドルの場合はピクセル単位で、プリンタの場合は 0.1mm 単位で設定してください。

【値の設定】 実行時

【値の参照】 不可

CharSet(イメージキットコントロール／PrintDraw プロパティ)

【機能】
フォントで使用する文字セットを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**CharSet** [*= short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.CharSet** [*= Smallint*]

【設定値】
フォントの文字セット(-1～255)
0～255 の値を設定した場合は、Windows で定義されている値と同じになります。

-1:ANSI 文字 or 日本語シフト JIS 文字 [ImageKit5 互換]
0:ANSI 文字(ANSI_CHARSET)
1:FontName と FontSize により選択(DEFAULT_CHARSET)
2:シンボル文字(SYMBOL_CHARSET)
128:日本語シフト JIS 文字(SHIFTJIS_CHARSET)
255:オペレーティングシステムに依存(OEM_CHARSET)

※()内の説明は WindowsAPI で使用する定数と同じ意味です。

【解説】
設定したフォントの文字セットは **DrawText**,**GetTextExtent**,**TextOut** メソッドで有効となります。

【値の設定】 実行時

【値の参照】 不可

Collate,PrintToFile(イメージキットコントロール／PrintDraw プロパティ)

【機能】
印刷ダイアログで処理する[部単位で印刷]と[ファイルへ出力]のチェックボックスを取得または設定します。

【書式】
※Collate にて説明(PrintToFile も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->PrintDraw->Collate [= *bool*]
(2)Delphi　*imagekitcontrolname*.PrintDraw.Collate [= *Boolean*]

【設定値】
Collate は[部単位で印刷]をチェックするかどうか(あるいはチェックされているかどうか)を示します。
PrintToFile は[ファイルへ出力]をチェックするかどうか(あるいはチェックされているかどうか)を示します。

値	説明
True	チェックする(されている)
False	チェックしない(されてない)

【解説】
PrintDialog メソッドで使用します。デフォルトは False です。
実行前にプロパティを設定するとその値で初期化し、実行後は参照できます。
ただし、PrintToFile プロパティについては Options プロパティに poPrintToFile 要素を指定しない場合、もしくは poDisablePrintToFile 要素を指定した場合は、[ファイルへ出力]は使用できません。
Collate プロパティについては、GetDevModeInfo メソッドや SetDevModeInfo メソッドでも使用します。

【値の設定】 実行時

【値の参照】 実行時

ColorMode(イメージキットコントロール／PrintDraw プロパティ)

【機能】
カラープリンタでカラーを使用するかモノクロを使用するかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**ColorMode** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.ColorMode** [= *Smallint*]

【設定値】

値	説明
1	モノクロ(DMCOLOR_MONOCHROME)
2	カラー(DMCOLOR_COLOR)

( )内の説明は WindowsAPI で使用する定数と同じ意味です。

【解説】
GetDevModeInfo メソッドを実行することによりプロパティ値が設定され(その後でプロパティ値を取得)、**SetDevModeInfo** メソッドを実行することによりプロパティ値を反映させます。
設定値はプリンタドライバに依存します。そのため、設定しても効果がまったくなかったり、異なる設定を行っても印刷結果が同じになることがあります。詳しくはプリンタドライバのマニュアルなどを参照してください。

【値の設定】 実行時

【値の参照】 実行時

Copies(イメージキットコントロール／PrintDraw プロパティ)

【機能】
印刷部数を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**Copies** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.Copies** [= *Smallint*]

【設定値】
印刷部数(1～)

【解説】
PrintDialog メソッドで使用します。実行前にプロパティを設定するとその値で初期化し、実行後は参照できます。
また、**GetDevModeInfo** メソッドや **SetDevModeInfo** メソッドでも使用します。

【値の設定】 実行時

【値の参照】 実行時

CustomPaperHeight,CustomPaperWidth(イメージキットコントロール／PrintDraw プロパティ)

【機能】
PaperSize プロパティが 256 の時の用紙の縦幅と横幅を表します。

【書式】
※CustomPaperHeight にて説明(CustomPaperWidth も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->PrintDraw->CustomPaperHeight [= *short*]
(2)Delphi　*imagekitcontrolname*.PrintDraw.CustomPaperHeight [= *Smallint*]

【設定値】
CustomPaperHeight は用紙の縦幅(0.1mm 単位)。
CustomPaperWidth は用紙の横幅(0.1mm 単位)。

【解説】
PaperSize プロパティが 256 の場合に有効です。
GetDevModeHandle,GetDevModeInfo メソッドを実行することによりプロパティ値が設定され(その後でプロパティ値を取得)、
SetDevModeInfo メソッドを実行することによりプロパティ値を反映させます。

【値の設定】 実行時

【値の参照】 実行時

DevMode(イメージキットコントロール／PrintDraw プロパティ)

【機能】
指定したプリンタの DEVMODE 構造体へのポインタのハンドルを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**DevMode** [= *NativeUInt*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**DevMode** [= *THandle*]

【設定値】
DEVMODE 構造体へのポインタのハンドル

【解説】
GetDevModeHandle メソッドや **PrintDialog** メソッドを実行すると、指定したプリンタの DEVMODE 構造体のハンドルを取得できます。**DevMode** プロパティが占有するメモリは、**ReleaseDevModeHandle** メソッドを実行すると解放されます(プロパティ値は 0 になります)。
DevMode プロパティは **Collate**,**ColorMode**,**Copies**,**CustomPaperHeight**,**CustomPaperWidth**,**Duplex**,**Orientation**,**PaperBin**,**PaperSize**,**XResolution**,**YResolution**,**Zoom** プロパティを管理します。
DEVMODE 構造体については WindowsAPI 関連の書籍などをご覧ください。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9 ActiveX/VCL との違い】
・値の設定が可能になりました。
・**PrintDialog** メソッド実行時にも値が更新されるようになりました。

DevNames(イメージキットコントロール／PrintDraw プロパティ)

【機能】
DEVNAMES 構造体へのポインタのハンドルを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**DevNames** [= *NativeUInt*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**DevNames** [= *THandle*]

【設定値】
DEVNAMES 構造体へのポインタのハンドル

【解説】
GetDevModeHandle メソッドや **PrintDialog** メソッドを実行すると、DEVMODE 構造体のハンドルに加えて DEVNAMES 構造体のハンドルも取得できます。**DevNames** プロパティが占有するメモリは、**ReleaseDevModeHandle** メソッドを実行すると解放されます(プロパティ値は 0 になります)。
DevNames プロパティが有効な場合、**GetDevNamesInfo** メソッドを実行することによりプリンタ名とポート名を取得できます。
DEVNAMES 構造体については WindowsAPI 関連の書籍などをご覧ください。

【値の設定】 実行時

【値の参照】 実行時

Direction(イメージキットコントロール／PrintDraw プロパティ)

【機能】
文字列の描画方向を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**Direction** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**Direction** [= *Smallint*]

【設定値】

値	説明
0	左から右(横書き)
1	右から左(横書き)
2	上から下(縦書き)
3	下から上(縦書き)

【解説】
RotateString プロパティが False の場合、**DrawText**,**TextOut** メソッドで有効です。

【値の設定】 実行時

【値の参照】 不可

Duplex(イメージキットコントロール／PrintDraw プロパティ)

【機能】
プリンタが両面印刷をサポートしている場合、両面印刷を行うかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**Duplex** [*= short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.Duplex** [*= Smallint*]

【設定値】

値	説明
1	両面印刷なし(DMDUP_SIMPLEX)
2	垂直(短い軸)方向の両面印刷(DMDUP_VERTICAL)
3	水平(長い軸)方向の両面印刷(DMDUP_HORIZONTAL)

( )内の説明は WindowsAPI で使用する定数と同じ意味です。

【解説】
GetDevModeHandle,**GetDevModeInfo** メソッドを実行することによりプロパティ値が設定され(その後でプロパティ値を取得)、**SetDevModeInfo** メソッドを実行することによりプロパティ値を反映させます。
設定値はプリンタドライバに依存します。そのため、設定しても効果がまったくなかったり、異なる設定を行っても印刷結果が同じになることがあります。詳しくはプリンタドライバのマニュアルなどを参照してください。

【値の設定】 実行時

【値の参照】 実行時

FontBold,FontItalic,FontStrikeOut,FontUnderline（イメージキットコントロール／PrintDraw プロパティ）

【機能】
フォントのスタイル・下線・打ち消し線を設定します。

【書式】
※FontBold にて説明（FontItalic,FontStrikeOut,FontUnderline も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->PrintDraw->FontBold [= *bool*]
(2)Delphi　*imagekitcontrolname*.PrintDraw.FontBold [= *Boolean*]

【設定値】
FontBold はフォントを太字にするかしないかの設定。
FontItalic はフォントを斜体にするかしないかの設定。
FontStrikeOut はテキストに打ち消し線を付加するかしないかの設定。
FontUnderline はテキストに下線を付加するかしないかの設定。

値	説明
True	する
False	しない

【解説】
設定したプロパティの情報は DrawString,DrawText,GetTextExtent,MeasureString,TextOut メソッドで有効になります。
FontStrikeOut,FontUnderline プロパティを True に設定した場合でも、CharAngle,Direction プロパティの値により無効になる場合があります。

【値の設定】 実行時

【値の参照】 不可

FontName,Text(イメージキットコントロール／PrintDraw プロパティ)

【機能】
テキストを描画する際のフォント名や文字列を設定します。

【書式】
※FontName にて説明(Text も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->PrintDraw->FontName [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.PrintDraw.FontName [= *string*]

【設定値】
FontName はフォントの名称(31 文字以内)。
Text は描画する文字列。

【解説】
設定したフォントと文字列は DrawString,DrawText,GetTextExtent,MeasureString,TextOut メソッドで有効となります。

【値の設定】 実行時

【値の参照】 不可

FontSize(イメージキットコントロール／PrintDraw プロパティ)

【機能】
テキストを描画する際のフォントのサイズを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**FontSize** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**FontSize** [= *Smallint*]

【設定値】
フォントのサイズ(ポイント)

【解説】
設定したフォントのサイズは **DrawString**,**DrawText**,**GetTextExtent**,**MeasureString**,**TextOut** メソッドで有効となります。
ただし、**DrawText** メソッドの場合は範囲内に収まるようにスケーリングされますので、設定したサイズがそのまま有効にならない場合があります。

【値の設定】 実行時

【値の参照】 不可

FromPage,MaxPage,MinPage,ToPage（イメージキットコントロール／PrintDraw プロパティ）

【機能】
印刷ダイアログで処理するページ数を取得または設定します。

【書式】
※**FromPage** にて説明（その他も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**FromPage** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**FromPage** [= *Smallint*]

【設定値】
FromPage は印刷開始ページ。
MaxPage は印刷最大ページ。
MinPage は印刷最小ページ。
ToPage は印刷終了ページ。

【解説】
PrintDialog メソッドで使用します。デフォルトは 0 です。
実行前にプロパティを設定するとその値で初期化し、実行後は指定したページ数が参照できます。ただし、**Options** プロパティにに poPageNums 要素を指定しない場合、[ページ指定]欄は使用できません。

【値の設定】 実行時

【値の参照】 実行時

Handle(イメージキットコントロール／PrintDraw プロパティ)

【機能】
プリンタのデバイスコンテキストを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw->Handle** [*= HDC*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.Handle** [*= HDC*]

【参照値】
プリンタのデバイスコンテキスト

【解説】
PrintCreateDC,**PrintDialog** メソッドを実行すると(成功した場合)、プリンタのデバイスコンテキストを取得できます。
取得したデバイスコンテキストは **PrintDeleteDC** メソッドで削除します。(プロパティ値は 0 もしくは NULL になります。)

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
プロパティの名称が **Hdc** から変更されました。

HCentering,VCentering(イメージキットコントロール／PrintDraw プロパティ)

【機能】
文字列のセンタリングを設定します。

【書式】
※HCentering にて説明(VCentering も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->PrintDraw->HCentering [= *short*]
(2)Delphi　*imagekitcontrolname*.PrintDraw.HCentering [= *Smallint*]

【設定値】
HCentering は水平センタリングを設定。
VCentering は垂直センタリングを設定。

値	説明
0	左(HCentering)、上(VCentering)
1	中央
2	右(HCentering)、下(VCentering)

【解説】
設定したプロパティの情報は DrawString,DrawText,MeasureString メソッドで有効になります。

【値の設定】 実行時

【値の参照】 不可

HotkeyPrefix(イメージキットコントロール／PrintDraw プロパティ)

【機能】
プリフィックス文字をどう扱うかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**HotkeyPrefix** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**HotkeyPrefix** [= *Smallint*]

【設定値】

値	説明
0	プリフィックスなし
1	プリフィックス表示
2	プリフィックス非表示(**DrawString** メソッドのみ)

【解説】
DrawString,**DrawText** メソッドで有効です。

例:
HotkeyPrefix = 0: &A は&A と描画されます。
HotkeyPrefix = 1: &AはAと描画されます。
HotkeyPrefix = 2: &A は A と描画されます。

【値の設定】 実行時

【値の参照】 不可

Options(イメージキットコントロール／PrintDraw プロパティ)

【機能】
印刷ダイアログの初期設定を行います。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw->Options** [= *TPrintDialogOptions*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.Options** [= *TPrintDialogOptions*]

【設定値】

値	説明
poDisablePrintToFile	[ファイルへ出力]チェックボックスを選択不可能(淡色表示)にする (poPrintToFile フラグが設定されている場合のみ適用可能)
poHelp	[ヘルプ]ボタンがダイアログに表示される
poPageNums	[ページ指定]ラジオボタンが使用可能
poPrintToFile	[ファイルへ出力]チェックボックスをダイアログに表示
poSelection	[選択した部分]ラジオボタンが使用可能
poWarning	インストールされていないプリンタに印刷ジョブを送ろうとした場合に警告メッセージを表示

【解説】
PrintDialog メソッドで使用します。
デフォルト値では、すべてのオプションフラグはオフです。
TPrintDialogOptions 型については Delphi や C++Builder のヘルプを参照してください。

例:
(1)C++Builder
```
// すべてのオプションをオフ
VImageKit1->PrintDraw->Options.Clear();
// ページ、選択した部分を有効
VImageKit1->PrintDraw->Options = VImageKit1->PrintDraw->Options << poPageNums << poSelection;
```
(2)Delphi
```
{ すべてのオプションをオフ }
VImageKit1.PrintDraw.Options := [];
{ ページ、選択した部分を有効 }
VImageKit1.PrintDraw.Options := [poPageNums, poSelection];
```

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
整数型から集合型に変更されました。

Orientation(イメージキットコントロール／PrintDraw プロパティ)

【機能】
印刷の向きを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**Orientation** [= *TPrinterOrientation*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.Orientation** [= *TPrinterOrientation*]

【設定値】

値	説明
poPortrait	縦
poLandscape	横

【解説】
GetDevModeHandle,**GetDevModeInfo** メソッドを実行することによりプロパティ値が設定され(その後でプロパティ値を取得)、**SetDevModeInfo** メソッドを実行することによりプロパティ値を反映させます。
設定値はプリンタドライバに依存します。そのため、設定しても効果がまったくなかったり、異なる設定を行っても印刷結果が同じになることがあります。詳しくはプリンタドライバのマニュアルなどを参照してください。
TPrinterOrientation 型については Delphi や C++Builder のヘルプを参照してください。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
整数型から列挙型に変更されました。

PaperBin(イメージキットコントロール／PrintDraw プロパティ)

【機能】
プリンタの用紙トレイを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**PaperBin** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**PaperBin** [= *Smallint*]

【設定値】

値	説明
1	上段用紙トレイ(DMBIN_UPPER)
2	下段用紙トレイ(DMBIN_LOWER)
3	中段用紙トレイ(DMBIN_MIDDLE)
4	手差し用紙フィーダ(DMBIN_MANUAL)
5	封筒フィーダ(DMBIN_ENVELOPE)
6	手差し封筒フィーダ(DMBIN_ENVMANUAL)
7	自動用紙トレイ選択(DMBIN_AUTO)
8	トラクタフィーダ(DMBIN_TRACTOR)
9	小判用紙ソース(DMBIN_SMALLFMT)
10	大判用紙ソース(DMBIN_LARGEFMT)
11	大容量の用紙トレイ(DMBIN_LARGECAPACITY)
14	用紙カセット(DMBIN_CASSETTE)

( )内の説明は WindowsAPI で使用する定数と同じ意味です。

【解説】
GetDevModeHandle,**GetDevModeInfo** メソッドを実行することによりプロパティ値が設定され(その後でプロパティ値を取得)、**SetDevModeInfo** メソッドを実行することによりプロパティ値を反映させます。
設定値はプリンタドライバに依存します。そのため、設定しても効果がまったくなかったり、異なる設定を行っても印刷結果が同じになることがあります。詳しくはプリンタドライバのマニュアルなどを参照してください。

【値の設定】 実行時

【値の参照】 実行時

PaperSize(イメージキットコントロール／PrintDraw プロパティ)

【機能】
プリンタの用紙サイズを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**PaperSize** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**PaperSize** [= *Smallint*]

【設定値】

値	説明
1	レター、8.5 x 11 インチ (DMPAPER_LETTER)
2	レター スモール、8.5 x 11 インチ (DMPAPER_LETTERSMALL)
3	タブロイド、11 x 17 インチ (DMPAPER_TABLOID)
4	レジャー、17 x 11 インチ (DMPAPER_LEDGER)
5	リーガル、8.5 x 14 インチ (DMPAPER_LEGAL)
6	ステートメント、5.5 x 8.5 インチ (DMPAPER_STATEMENT)
7	エグゼクティブ、7.25 x 10.5 インチ (DMPAPER_EXECUTIVE)
8	A3、297 x 420 mm (DMPAPER_A3)
9	A4、210 x 297 mm (DMPAPER_A4)
10	A4 Small、210 x 297 mm (DMPAPER_A4SMALL)
11	A5、148 x 210 mm (DMPAPER_A5)
12	B4、250 x 354 mm (DMPAPER_B4)
13	B5、182 x 257 mm (DMPAPER_B5)
256	ユーザ定義サイズ(DMPAPER_USER)

( )内の説明は WindowsAPI で使用する定数と同じ意味です。

【解説】
GetDevModeHandle,**GetDevModeInfo** メソッドを実行することによりプロパティ値が設定され(その後でプロパティ値を取得)、**SetDevModeInfo** メソッドを実行することによりプロパティ値を反映させます。
14 以降の値も取得および設定可能です。詳しくはご利用のコンテナの WindowsAPI に関連する部分を参考してください。
設定値はプリンタドライバに依存します。そのため、設定しても効果がまったくなかったり、異なる設定を行っても印刷結果が同じになることがあります。詳しくはプリンタドライバのマニュアルなどを参照してください。

【値の設定】 実行時

【値の参照】 実行時

PenMode(イメージキットコントロール／PrintDraw プロパティ)

【機能】
描かれるペンの色をキャンバス上の色とどのように対応させるかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**PenMode** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.PenMode** [= *Smallint*]

【設定値】

値	説明
1	黒(R2_BLACK)
2	ペン色と画面色の組み合わせの反転(R2_NOTMERGEPEN)
3	画面色とペン反転色のどちらにも共通な色の組み合わせ(R2_MASKNOTPEN)
4	ペンの反転色(R2_NOTCOPYPEN)
5	ペン色と画面反転色のどちらにも共通な色の組み合わせ(R2_MASKPENNOT)
6	画面色の反転色(R2_NOT)
7	ペン色または画面色の(両方ではなく)どちらかの色の組み合わせ(R2_XORPEN)
8	ペン色と画面色のどちらにも共通な色の組み合わせの反転(R2_NOTMASKPEN)
9	ペン色と画面色のどちらにも共通な色の組み合わせ(R2_MASKPEN)
10	ペン色または画面色の(両方ではなく)どちらかの色の組み合わせの反転(R2_NOTXORPEN)
11	変化なし(R2_NOP)
12	画面色とペン反転色の組み合わせ(R2_MERGENOTPEN)
13	**PenColor** で指定したペン色(R2_COPYPEN)、デフォルト値
14	ペン色と画面反転色の組み合わせ(R2_MERGEPENNOT)
15	ペン色と画面色の組み合わせ(R2_MERGEPEN)
16	白(R2_WHITE)

( )内の説明は WindowsAPI で使用する定数と同じ意味です。

定数(vikPenModeBlack = 1, vikPenModeNotMergePen = 2, vikPenModeMaskNotPen = 3, vikPenModeNotCopyPen = 4, vikPenModeMaskPenNot = 5, vikPenModeNot = 6, vikPenModeXorPen = 7, vikPenModeNotMaskPen = 8, vikPenModeMaskPen = 9, vikPenModeNotXorPen = 10, vikPenModeNop = 11, vikPenModeMergeNotPen = 12, vikPenModeCopyPen = 13, vikPenModeMergePenNot = 14, vikPenModeMergePen = 15, vikPenModeWhite = 16)を使用することも可能です。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
定数を使用する場合、識別子の先頭に v が付加されました(ActiveX は ikPenModeBlack, ikPenModeNotMergePen, ikPenModeMaskNotPen, ikPenModeNotCopyPen, ikPenModeMaskPenNot, ikPenModeNot, ikPenModeXorPen, ikPenModeNotMaskPen, ikPenModeMaskPen, ikPenModeNotXorPen, ikPenModeNop, ikPenModeMergeNotPen, ikPenModeCopyPen, ikPenModeMergePenNot, ikPenModeMergePen, ikPenModeWhite)。

PenStyle(イメージキットコントロール／PrintDraw プロパティ)

【機能】
描画するペンのスタイルを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**PenStyle** [= *TVIkPenStyle*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**PenStyle** [= *TVIkPenStyle*]

【TVIkPenStyle 型】
ユニット
IkInit
type
TVIkPenStyle = (vikPenNull, vikPenSolid, vikPenDash, vikPenDot, vikPenDashDot, vikPenDashDotDot, vikPenInsideFrame);

【設定値】

値	説明
vikPenNull	線は描画されない(PS_NULL)
vikPenSolid	実線(PS_SOLID)
vikPenDash	破線(PS_DASH)
vikPenDot	点線(PS_DOT)
vikPenDashDot	一点鎖線(PS_DASHDOT)
vikPenDashDotDot	二点鎖線(PS_DASHDOTDOT)
vikPenInsideFrame	実線(PS_INSIDEFRAME)

( )内の説明は WindowsAPI で使用する定数と同じ意味です。

【解説】
Line,**Rectangle** メソッド以外で図形を描画する場合に、**PenWidth** プロパティに 1 以上(ピクセルで与えた場合、0.1mm の場合もピクセルに換算して判定)かつ **PenStyle** プロパティに vikPenNull,vikPenSolid,vikPenInsideFrame 以外を設定した場合には **PenWidth** プロパティを強制的に 1 とします。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikPenNull, ikPenSolid, ikPenDash, ikPenDot, ikPenDashDot, ikPenDashDotDot, ikPenInsideFrame)。

PenWidth(イメージキットコントロール／PrintDraw プロパティ)

【機能】
描画するペンの幅を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**PenWidth** [= *int*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**PenWidth** [= *Integer*]

【設定値】
ペンの幅(ピクセルもしくは 0.1mm 単位)

【解説】
描画対象がスクリーンとメモリハンドルの場合はピクセル単位で、プリンタの場合は 0.1mm 単位で設定してください。

【値の設定】 実行時

【値の参照】 不可

PortName(イメージキットコントロール／PrintDraw プロパティ)

【機能】
プリンタのポート名を示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**PortName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**PortName** [= *string*]

【参照値】
ポート名。

【解説】
GetDevModeHandle,**GetDevNamesInfo**,**GetPrinterPort** メソッドを実行するとポート名が設定されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9 ActiveX/VCL との違い】
GetDevModeHandle,**GetDevNamesInfo** メソッド実行後にもポート名が設定されます。

Ports(イメージキットコントロール／PrintDraw プロパティ)

【機能】
EnumPorts メソッドで取得されるポート名のリストを示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->PrintDraw->Ports [= *TStrings**]
(2)Delphi　*imagekitcontrolname*.PrintDraw.Ports [= *TStrings*]

【参照値】
ポート名のリスト。

【解説】
EnumPorts メソッドが成功した場合、リスト文字列が Ports プロパティに設定されます。
ポートの数は Ports.Count で取得できます。

【値の設定】 不可

【値の参照】 実行時

PrinterName(イメージキットコントロール／PrintDraw プロパティ)

【機能】
プリンタ名を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**PrinterName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**PrinterName** [= *string*]

【設定値】
プリンタ名

【解説】
GetDevModeHandle,PrintCreateDC メソッドなどを使用する際に設定します。
プリンタ名は **EnumPrinters** メソッドで取得した名称を設定してください。
また、**GetDevModeHandle,GetDevNamesInfo** メソッドを実行し成功するとプリンタ名が更新されます。

例:VImageKit1.PrintDraw.PrinterName := 'EPSON LP-8200C';

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9 ActiveX/VCL との違い】
GetDevModeHandle,GetDevNamesInfo メソッドを実行し成功するとプリンタ名が更新されます。

PrinterIndex,Printers（イメージキットコントロール／PrintDraw プロパティ）

【機能】
EnumPrinters メソッドで取得されるプリンタ名のリストを示します。

【書式】
※PrinterIndex
(1)C++Builder　*imagekitcontrolname*->**PrinterIndex** [*= int*]
(2)Delphi　*imagekitcontrolname*.**PrinterIndex** [*= Integer*]

※Printers
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**Printers** [*= TStrings**]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**Printers** [*= TStrings*]

【参照値】
PrinterIndex はデフォルトプリンタのインデックス。
Printers はプリンタ名のリスト。

【解説】
EnumPrinters メソッドが成功した場合、リスト文字列が Printers プロパティに、デフォルトプリンタのインデックスが PrinterIndex プロパティに設定されます。
インストールされているプリンタの数は Printers.Count で取得できます。
PrinterIndex プロパティが-1 の場合は EnumPrinters メソッドが失敗もしくはメソッドを実行していない状態を示し、0 以上の場合は Printers.Strins プロパティにアクセスする配列の添字として使用可能です。

デフォルトプリンタに位置付ける例：
(1)C++Builder
```
VImageKit1->PrintDraw->EnumPrinters();
if (VImageKit1->PrintDraw->Printers->Count > 0)
{
   ComboBox1->Items = VImageKit1->PrintDraw->Printers;
   ComboBox1->ItemIndex = VImageKit1->PrintDraw->PrinterIndex;
}
```
(2)Delphi
```
VImageKit1.PrintDraw.EnumPrinters();
if (VImageKit1.PrintDraw.Printers.Count > 0) then
begin
   ComboBox1.Items := VImageKit1.PrintDraw.Printers;
   ComboBox1.ItemIndex := VImageKit1.PrintDraw.PrinterIndex;
end;
```

【値の設定】 不可

【値の参照】 実行時

PrintFileName(イメージキットコントロール／PrintDraw プロパティ)

【機能】
プリンタの情報を保存するファイル名を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**PrintFileName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**PrintFileName** [= *string*]

【設定値】
ファイル名

【解説】
SaveDevModeHandle もしくは **SavePrinterInfo** メソッドで保存するファイル名を設定します(誤動作を防ぐためにフルパスで設定してください)。

拡張機能として **PrintCreateDC** メソッドを実行する前に、**PrintFileName** プロパティにプリンタの設定情報を保存したファイル名にプラスして印刷部数、印刷の向き、用紙サイズを順番に設定すると、保存した項目の該当する部分を変更して印刷することができます(後ろの項目は省略可、順番を入れ替えることは不可)。
設定する場合はそれぞれの項目をセミコロン(;)で区切ります。
設定ファイルに保存された情報を有効にしたい場合は、それぞれの該当する項目に 0 を設定します。

印刷の向き:()内の説明は WindowsAPI で使用する定数と同じ意味です。
1:縦(DMORIENT_PORTRAIT)　2:横(DMORIENT_LANDSCAPE)
用紙サイズ:()内の説明は WindowsAPI で使用する定数と同じ意味です。
8:A3(DMPAPER_A3)　9:A4(DMPAPER_A4)
12:B4(DMPAPER_B4)　13:B5(DMPAPER_B5)
上記以外の用紙サイズについては、ご利用のコンテナの WindowsAPI に関連する部分を参考に設定してください。

設定ファイルを"IkPrint.Ik"とした場合(Delphi)
1)通常の場合
imagekitcontrolname.**PrintFileName** := 'C:¥Test¥IkPrint.Ik';
2)拡張機能を使用した場合
A)印刷部数を 3、印刷の向きを横、用紙サイズを B5 にする
imagekitcontrolname.**PrintFileName** := 'C:¥Test¥IkPrint.Ik;3;2;13';
B)印刷部数と用紙サイズを設定ファイルから読み出し、印刷の向きを横にする
imagekitcontrolname.**PrintFileName** := 'C:¥Test¥IkPrint.Ik;0;2;0';

【値の設定】 実行時

【値の参照】 実行時

PrintRange(イメージキットコントロール／PrintDraw プロパティ)

【機能】
印刷ダイアログの印刷範囲の種類を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**PrintRange** [= *TPrintRange*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**PrintRange** [= *TPrintRange*]

【設定値】

値	説明
prAllPages	[すべて]ラジオボタンが選択される
prSelection	[選択した部分]ラジオボタンが選択される
prPageNums	[ページ指定]ラジオボタンが選択される

【解説】
PrintDialog メソッドで使用します。実行前にプロパティを設定するとその値で初期化し、実行後は選択された印刷範囲を参照できます。ただし、**Options** プロパティの設定によっては prSelection,prPageNums は有効にならない場合があります。
TPrintRange 型については Delphi や C++Builder のヘルプを参照してください。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
整数型から列挙型に変更されました。

RotateString(イメージキットコントロール／PrintDraw プロパティ)

【機能】

Text プロパティで設定された文字列を文字単位で回転するか文字列単位で回転するかを設定します。

【書式】

(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**RotateString** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**RotateString** [= *Boolean*]

【設定値】

値	説明
True	文字列単位
False	文字単位

【解説】

GetTextExtent,**TextOut** メソッドで使用します。False の場合は **CharAngle** プロパティは 0,90,180,270 のみ有効となります。

【値の設定】 実行時

【値の参照】 不可

Transparent(イメージキットコントロール／PrintDraw プロパティ)

【機能】
テキストやハッチブラシの背景および線と線の間隔を描画する時の透過を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**Transparent** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.Transparent** [= *Boolean*]

【設定値】

値	説明
True	透過有効
False	透過無効

【解説】
Transparent プロパティを True に設定した場合は、**BackColor,TextColor2** プロパティは無視されます。
DrawString メソッドでは **Transparent** プロパティの設定に関わらずテキストの背景は透過されます。

【値の設定】 実行時

【値の参照】 不可

XResolution,YResolution(イメージキットコントロール／PrintDraw プロパティ)

【機能】
プリンタの X 方向と Y 方向の解像度を取得または設定します。

【書式】
※XResolution にて説明(YResolution も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->PrintDraw->XResolution [= *short*]
(2)Delphi　*imagekitcontrolname*.PrintDraw.XResolution [= *Smallint*]

【設定値】
正の値: 1 インチあたりのドット数(DPI)

値	説明
-1	ドラフトの印刷解像度(DMRES_DRAFT)
-2	低解像度(DMRES_LOW)
-3	中解像度(DMRES_MEDIUM)
-4	高解像度(DMRES_HIGH)

( )内の説明は WindowsAPI で使用する定数と同じ意味です。

【解説】
XResolution プロパティに 0 以下の値を設定すると YResolution プロパティの値は無視されます。
GetDevModeHandle,GetDevModeInfo メソッドを実行することによりプロパティ値が設定され(その後でプロパティ値を取得)、SetDevModeInfo メソッドを実行することによりプロパティ値を反映させます。
設定値はプリンタドライバに依存します。そのため、設定しても効果がまったくなかったり、異なる設定を行っても印刷結果が同じになることがあります。詳しくはプリンタドライバのマニュアルなどを参照してください。
また、PrintCreateDC,PrintDialog メソッドを実行し、デバイスコンテキストが作成されるとそれに応じた解像度が設定されます。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9 ActiveX/VCL との違い】
PrintCreateDC,PrintDialog メソッドを実行し、デバイスコンテキストが作成されるとそれに応じた解像度が設定されます。

Zoom(イメージキットコントロール／PrintDraw プロパティ)

【機能】
印刷結果のスケール時の百分率を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**Zoom** [= *short*]
(2)Delphi　*imagekitcontrolname*.**PrintDraw.Zoom** [= *Smallint*]

【設定値】
スケール時の百分率(0～100)

【解説】
GetDevModeHandle,**GetDevModeInfo** メソッドを実行することによりプロパティ値が設定され(その後でプロパティ値を取得)、**SetDevModeInfo** メソッドを実行することによりプロパティ値を反映させます。
設定値はプリンタドライバに依存します。そのため、設定しても効果がまったくなかったり、異なる設定を行っても印刷結果が同じになることがあります。詳しくはプリンタドライバのマニュアルなどを参照してください。

【値の設定】 実行時

【値の参照】 実行時

Arc（イメージキットコントロール／PrintDraw メソッド）

【機能】
弧を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Arc**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Arc**(NativeUInt DeviceValue, const TRect &ARect, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Arc**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Arc**(void * DeviceValue, const TRect &ARect, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Arc**(DeviceValue: THandle; ALeft, ATop, ARight, ABottom, X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Arc**(DeviceValue: THandle; const ARect: TRect; X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Arc**(DeviceValue: Pointer; ALeft, ATop, ARight, ABottom, X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Arc**(DeviceValue: Pointer; const ARect: TRect; X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
X1,Y1	弧の始点の x,y 座標
X2,Y2	弧の終点の x,y 座標
DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ALeft,ATop,ARight,ABottom もしくは ARect で指定した矩形に含まれる楕円弧を 2 点間で描画します。始点と終点が弧の上にある必要はなく、指定の点から境界矩形までの中心までの直線が計算され、弧とその直線との交点が用いられます。
（描画対象はスクリーン、プリンタ、メモリハンドル）
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。
BackColor プロパティは **Transparent** プロパティが False でペンが実線以外の場合に有効です。

ALeft,ATop,ARight,ABottom(もしくはARect),X1,Y1,X2,Y2 をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect),X1,Y1,X2,Y2 を 0.1mm単位として扱う場合

DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました（ActiveX は ikScreen, ikPrinter, ikMemoryHandle）。
・TRect 型を渡すメソッドが実装されました。

Chord(イメージキットコントロール／PrintDraw メソッド)

【機能】
描画先オブジェクトに弓形を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Chord**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Chord**(NativeUInt DeviceValue, const TRect &ARect, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Chord**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Chord**(void * DeviceValue, const TRect &ARect, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Chord**(DeviceValue: THandle; ALeft, ATop, ARight, ABottom, X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Chord**(DeviceValue: THandle; const ARect: TRect; X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Chord**(DeviceValue: Pointer; ALeft, ATop, ARight, ABottom, X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Chord**(DeviceValue: Pointer; const ARect: TRect; X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル(DeviceMode による)
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
X1,Y1	最初の径の端点の x,y 座標
X2,Y2	次の径の端点の x,y 座標
DeviceMode	描画対象(vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル)

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ALeft,ATop,ARight,ABottom もしくは ARect で指定した矩形に含まれる楕円弧を(x1,y1),(x2,y2)の間を通る線によって 2 等分された領域を描画します。アウトラインは **PenStyle** プロパティの値で描画され、内部は **BrushStyle** プロパティの値で塗りつぶされます。
(対象はスクリーン、プリンタ、メモリハンドル)
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**BrushStyle**,**BrushColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。**BackColor** プロパティは **Transparent** プロパティが False でペンが実線以外、あるいはブラシがハッチパターンの場合に有効です。

ALeft,ATop,ARight,ABottom(もしくはARect),X1,Y1,X2,Y2 をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect),X1,Y1,X2,Y2 を 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。
・TRect 型を渡すメソッドが実装されました。

ClearProperty(イメージキットコントロール／PrintDraw メソッド)

【機能】
イメージキットコントロールの **PrintDraw** プロパティを初期化します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**PrintDraw**->**ClearProperty**()
(2)Delphi　*imagekitcontrolname*.**PrintDraw**.**ClearProperty**

【引数】
ありません。

【戻り値】
ありません。

【解説】
値を設定可能なプロパティを初期化します(0 もしくは False)。**CharSet** プロパティは-1 となります。

DrawFocusRect(イメージキットコントロール／PrintDraw メソッド)

【機能】
画面にフォーカスを示す矩形を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawFocusRect**(HDC DC, int ALeft, int ATop, int ARight, int ABottom, TVIkDeviceUnitMode UnitMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawFocusRect**(HDC DC, const TRect &ARect, TVIkDeviceUnitMode UnitMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawFocusRect**(DC: HDC; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; UnitMode: TVIkDeviceUnitMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawFocusRect**(DC: HDC; const ARect: TRect; UnitMode: TVIkDeviceUnitMode)

【TVIkDeviceUnitMode 型】
ユニット
ImageKit
type
TVIkDeviceUnitMode = (vikPrinterPixel, vikPrinterMM);

【引数】

名称	内容
DC	デバイスコンテキスト
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
UnitMode	ALeft,ATop,ARight,ABottom もしくは ARect の単位(vikPrinterPixel:ピクセル、vikPrinterMM:0.1mm)

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
XOR 機能を使用して矩形を描画するため、同じ矩形を引数にして 2 回呼び出すと画面から矩形が消去されます。描画した矩形を含む領域をスクロールするには、再度矩形を描画して画面から消去し、領域をスクロールした後にもう一度新しい位置に同じ矩形を描画します。
描画するには **BrushStyle**,**BrushColor** プロパティに値を設定する必要があります。

ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合
UnitMode が vikPrinterPixel
ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合
UnitMode が vikPrinterMM

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikPrinterPixel, ikPrinterMM)。
・TRect 型を渡すメソッドが実装されました。

【ImageKit7 VCL との違い】
C++Builder では引数 DC の unsigned 型が、Delphi では Pointer 型のメソッドが削除されました。

DrawString(イメージキットコントロール／PrintDraw メソッド)

【機能】
GDI+の機能を利用して描画先オブジェクトにテキストを描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawString**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, short BrushType, short Style, NativeUInt ImageHandle, short TextRenderingHint, int FormatFlags, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawString**(NativeUInt DeviceValue, const TRect &ARect, short BrushType, short Style, NativeUInt ImageHandle, short TextRenderingHint, int FormatFlags, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawString**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, short BrushType, short Style, NativeUInt ImageHandle, short TextRenderingHint, int FormatFlags, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawString**(void * DeviceValue, const TRect &ARect, short BrushType, short Style, NativeUInt ImageHandle, short TextRenderingHint, int FormatFlags, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawString**(DeviceValue: THandle; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; BrushType: Smallint; Style: Smallint; ImageHandle: THandle; TextRenderingHint: Smallint; FormatFlags: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawString**(DeviceValue: THandle; const ARect: TRect; BrushType: Smallint; Style: Smallint; ImageHandle: THandle; TextRenderingHint: Smallint; FormatFlags: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawString**(DeviceValue: Pointer; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; BrushType: Smallint; Style: Smallint; ImageHandle: THandle; TextRenderingHint: Smallint; FormatFlags: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawString**(DeviceValue: Pointer; const ARect: TRect; BrushType: Smallint; Style: Smallint; ImageHandle: THandle; TextRenderingHint: Smallint; FormatFlags: Integer; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル(DeviceMode による)
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
BrushType	ブラシの種類(0:ソリッド、1:ハッチ、2:テクスチャ、4:グラデーション)
Style	ブラシ毎のスタイル(BrushType が 0 以外の場合に有効)
ImageHandle	テクスチャブラシ(BrushType=4)の時に使用するラスタイメージのメモリハンドル
TextRenderingHint	テキストのレンダリングモード(デフォルトは 0)
FormatFlags	文字列の書式情報(デフォルトは 0)
DeviceMode	描画対象(vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル)

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】

ARight あるいは ABottom を 0 に設定すると ALeft と ATop を始点としてテキストを描画します（ARect を使用する場合は Right と Bottom を 0 に、Left と Top に始点を設定）。それ以外の場合は ALeft,ATop,ARight,ABottom もしくは ARect で囲まれる矩形の内部にテキストが描画され、内部に収まらないテキストは切り捨てられます。
（対象はスクリーン、プリンタ、メモリハンドル）
描画するには **TextColor1,Alpha1,TextColor2,Alpha2,FontName,FontSize,CharAngle,HCentering,VCentering, HotkeyPrefix,Text** プロパティに値を設定する必要があります。
HotkeyPrefixプロパティに 1 以外を設定するとプリフィックス文字の処理を行いません（&Aは<u>A</u>にはならず、&AもしくはAと描画されます）。
TextColor1,Alpha1 プロパティは「ソリッドブラシ：ブラシの色」「ハッチブラシ：描画される線の色」「グラデーションブラシ：線形グラデーションの開始色」を表します。**TextColor2,Alpha2** プロパティは「ハッチブラシ：線間の領域の色」「グラデーションブラシ：線形グラデーションの終了色」を表します。
グラデーションブラシ（BrushType=4）を使用する場合は、ALeft,ATop,ARight,ABottom もしくは ARect に矩形領域を設定してください。

<u>ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合</u>
DeviceMode が vikScreen もしくは vikMemoryHandle
<u>ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合</u>
DeviceMode が vikPrinter

「ブラシ毎のスタイル（Style に設定する値）」
ハッチブラシの場合：

値	説明
0	水平線のパターン。
1	垂直線のパターン。
2	左上から右下への対角線のパターン。
3	右上から左下への対角線のパターン。
4	交差する水平および垂直の線を指定します。
5	交差する右上がりと左上がりの対角線を指定します。線はアンチエイリアス処理されます。
6	5%のハッチを指定します。前景色の背景色に対する割合は、5: 100 です。
7	10%のハッチを指定します。前景色の背景色に対する割合は、10: 100 です。
8	20%のハッチを指定します。前景色の背景色に対する割合は、20: 100 です。
9	25%のハッチを指定します。前景色の背景色に対する割合は、25: 100 です。
10	30%のハッチを指定します。前景色の背景色に対する割合は、30: 100 です。
11	40%のハッチを指定します。前景色の背景色に対する割合は、40: 100 です。
12	50%のハッチを指定します。前景色の背景色に対する割合は、50: 100 です。
13	60%のハッチを指定します。前景色の背景色に対する割合は、60: 100 です。
14	70%のハッチを指定します。前景色の背景色に対する割合は、70: 100 です。
15	75%のハッチを指定します。前景色の背景色に対する割合は、75: 100 です。
16	80%のハッチを指定します。前景色の背景色に対する割合は、80: 100 です。
17	90%のハッチを指定します。前景色の背景色に対する割合は、90: 100 です。
18	上の点から下の点へ右に傾斜し、2 よりも間隔が 50%狭く、アンチエイリアス処理されない対角線。
19	上の点から下の点へ左に傾斜し、3 よりも間隔が 50%狭く、アンチエイリアス処理されない対角線。
20	上の点から下の点へ右に傾斜し、2 よりも間隔が 50%狭く、幅が 2 倍の対角線。このハッチパターンはアンチエイリアス処理されません。
21	上の点から下の点へ左に傾斜し、3 よりも間隔が 50%狭く、幅が 2 倍で、線がアンチエイリアス処理されない対角線。
22	上の点から下の点へ右に傾斜し、2 と間隔が等しく、幅が 3 倍で、アンチエイリアス処理されない対角線。
23	上の点から下の点へ左に傾斜し、3 と間隔が等しく、幅が 3 倍で、アンチエイリアス処理されない対角線。
24	間隔が 1 よりも 50%狭い垂直線を指定します。
25	間隔が 0 よりも 50%狭い水平線を指定します。
26	間隔が 1 よりも 75%狭い(または 24 よりも 25%狭い)垂直線を指定します。
27	間隔が 0 よりも 75%狭い(または 25 よりも 25%狭い)水平線を指定します。
28	1 よりも間隔が 50%狭く、幅が 2 倍の垂直線を指定します。
29	0 よりも間隔が 50%狭く、幅が 2 倍の水平線を指定します。
30	上の点から下の点へ右に傾斜した、破線の対角線を指定します。
31	上の点から下の点へ左に傾斜した、破線の対角線を指定します。

32	破線の水平線を指定します。
33	破線の垂直線を指定します。
34	紙吹雪のように見えるハッチを指定します。
35	34 よりも大きいピースから構成される、紙吹雪のように見えるハッチを指定します。
36	ジグザグに構成された水平線を指定します。
37	ティルダで構成された水平線を指定します。
38	上の点から下の点へ左に傾斜した、積み重ねたレンガ状のハッチを指定します。
39	レンガを水平に積み上げたように見えるハッチを指定します。
40	織物のように見えるハッチを指定します。
41	格子柄の生地のように見えるハッチを指定します。
42	芝生のように見えるハッチを指定します。
43	それぞれがドットで構成されて交差する、水平線と垂直線を指定します。
44	それぞれがドットで構成されて交差する、右上がりと左上がりの対角線を指定します。
45	上の点から下の点へ右に傾斜した、対角線状に積み重ねた板屋根のように見えるハッチを指定します。
46	四目格子のように見えるハッチを指定します。
47	交互に並べた球体のように見えるハッチを指定します。
48	間隔が 4 よりも 50%狭い、交差する水平線と垂直線を指定します。
49	チェッカーボードのように見えるハッチを指定します。
50	49 の 2 倍のサイズの正方形による、チェッカーボードのように見えるハッチを指定します。
51	交差する右上がりと左上がりの線で、アンチエイリアス処理されない対角線を指定します。
52	斜めに置かれたチェッカーボードのように見えるハッチを指定します。

テクスチャブラシの場合:

値	説明
0	グラデーションまたはテクスチャを並べて表示します。
1	テクスチャまたはグラデーションを水平方向に反転し、それを並べて表示します。
2	テクスチャまたはグラデーションを垂直方向に反転し、それを並べて表示します。
3	テクスチャまたはグラデーションを水平および垂直方向に反転し、それを並べて表示します。
4	テクスチャまたはグラデーションをオブジェクトの端に揃えます。

グラデーションブラシの場合:

値	説明
0	左から右へのグラデーションを指定します。
1	上から下へのグラデーションを指定します。
2	左上から右下へのグラデーションを指定します。
3	右上から左下へのグラデーションを指定します。

「テキストのレンダリングモード」

値	説明
0	グリフビットマップを使用し、システム既定のレンダリングヒントで各文字を描画することを指定します。
1	グリフビットマップを使用して、各文字を描画することを指定します。ヒンティングを使用して、文字のステム部分と曲線部分の見た目を向上します。
2	グリフビットマップを使用して、各文字を描画することを指定します。ヒンティングは使用されません。
3	アンチエイリアス処理されたグリフビットマップを使用して、ヒンティングありで各文字を描画することを指定します。アンチエイリアスによってより高い品質が得られますが、パフォーマンスは大きく低下します。
4	アンチエイリアス処理されたグリフビットマップを使用して、ヒンティングなしに各文字を描画することを指定します。アンチエイリアスによって品質が向上します。ヒンティングがオフにされるため、ステム幅の違いが目立ちます。
5	グリフ CT ビットマップを使用して、ヒンティングありで各文字を描画することを指定します。最高の品質設定です。ClearType テキストフォント機能を利用するときに使用します。

「文字列の書式情報」
設定値は次の値の論理和となります。

値	説明

0x00000001	テキストの方向を右から左に指定します。
0x00000002	テキストが縦書きになるよう指定します。
0x00000004	グリフが外接する四角形にかからないよう指定します。既定では、端に表示する必要がある場合、一部のグリフが若干四角形にかかるよう設定されます。
0x00000020	左から右を指示するマークなどの制御文字をグリフで表現します。
0x00000400	フォールバックを無効にして、要求されたフォントでサポートされていない文字のフォントを切り替えます。欠落文字は、グリフの欠落したフォント(通常は空白の正方形)で表示されます。
0x00000800	既定では、各行末の空白が除外されます。各行末の空白を計測に含める場合はこの値を設定します。
0x00001000	四角形内の書式指定時に、行間のテキストのラップを無効にします。この値は、四角形ではなく点が渡された場合、または長さゼロの行の四角形が指定された場合に暗黙的に指定されます。
0x00002000	書式指定用の四角形には、完全な直線だけがレイアウトされます。既定では、クリッピングの結果、テキストの末尾が表示された状態、または行が表示されなくなった状態のうち、いずれか早い方の状態になるまでレイアウトが継続します。既定の設定では、行高さの整数倍でない書式指定用四角形を用いた場合は、なるまでレイアウトが継続します。既定の設定では、行高さの整数倍でない書式指定用四角形を用いた場合は、最後の行の一部が隠れることがあります。必ず行全体が表示されるようにするには、この値を指定した上で、少なくとも 1 つの行と高さが同じの書式指定用の四角形をご使用ください。
0x00004000	論理グリフの突出部と書式指定用の四角形からはみ出すラップされていないテキストを表示できます。既定では、書式指定用の四角形からはみ出たテキストとグリフ部はすべてクリップされます。

定数(vikDirectionRightToLeft = 0x00000001, vikDirectionVertical = 0x00000002, vikNoFitBlackBox = 0x00000004, vikDisplayFormatControl = 0x00000020, vikNoFontFallback = 0x00000400, ikMeasureTrailingSpaces = 0x00000800, vikNoWrap = 0x00001000, vikLineLimit = 0x00002000, vikNoClip = 0x00004000)を使用することも可能です。

※16 進表記のため、Delphi は 0x を$に置き換えてください。

コード例:

(1)C++Builder

```
float Str_Width, Str_Height;
int charactersFitted, linesFilled, ARight, ABottom;

VImageKit1->PrintDraw->ClearProperty();
VImageKit1->PrintDraw->FontSize = 20;
VImageKit1->PrintDraw->FontName = "MS ゴシック";
VImageKit1->PrintDraw->HotkeyPrefix = 0;
VImageKit1->PrintDraw->Text = "テキスト";
VImageKit1->PrintDraw->MeasureString(DC, 0, 0, Str_Width, Str_Height, charactersFitted, linesFilled, 0,
vikPrinterPixel);

ARight = 10 + (int)Str_Width - 1;
ABottom = 10 + (int)Str_Height - 1;
VImageKit1->PrintDraw->CharAngle = 0;
VImageKit1->PrintDraw->TextColor1 = clRed;
VImageKit1->PrintDraw->Alpha1 = 255;
VImageKit1->PrintDraw->TextColor2 = clWhite;
VImageKit1->PrintDraw->Alpha2 = 255;
//グラデーションブラシを使用
VImageKit1->PrintDraw->DrawString(DC, 10, 10, ARight, ABottom, 4, 0, 0, 0, 0,
TVIkOutPutDeviceMode(vikPrinterPixel));
```

(2)Delphi

```
Str_Width, Str_Height: Single;
CharactersFitted, linesFilled, TxRight, TxBottom: Integer;

VImageKit1.PrintDraw.ClearProperty();
VImageKit1.PrintDraw.FontSize := 20;
VImageKit1.PrintDraw.FontName := 'MS ゴシック';
VImageKit1.PrintDraw.HotkeyPrefix := 0;
VImageKit1.PrintDraw.Text := 'テキスト';
VImageKit1.PrintDraw.MeasureString(DC, 0, 0, Str_Width, Str_Height, charactersFitted, linesFilled, 0,
vikPrinterPixel);
```

```
TxRight := 10 + Trunc(Str_Width) - 1;
TxBottom := 10 + Trunc(Str_Height) - 1;
VImageKit1.PrintDraw.CharAngle := 0;
VImageKit1.PrintDraw.TextColor1 := clRed;
VImageKit1.PrintDraw.Alpha1 := 255;
VImageKit1.PrintDraw.TextColor2 := clWhite;
VImageKit1.PrintDraw.Alpha2 := 255;
//グラデーションブラシを使用
VImageKit1.PrintDraw.DrawString(DC, 10, 10, TxRight, TxBottom, 4, 0, 0, 0, 0,
TVIkOutPutDeviceMode(vikPrinterPixel));
```

【ImageKit7/8/9/10 ActiveX との違い】

・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。

・定数を使用する場合、識別子の先頭に v が付加されました(ActiveX は ikDirectionRightToLeft, ikDirectionVertical, ikNoFitBlackBox, ikDisplayFormatControl, ikNoFontFallback, ikMeasureTrailingSpaces, ikNoWrap, ikLineLimit, ikNoClip)。

・TRect 型を渡すメソッドが実装されました。

DrawText（イメージキットコントロール／PrintDraw メソッド）

【機能】
描画先オブジェクトにテキストを描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawText**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, bool EnableFontScale, bool Clip, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawText**(NativeUInt DeviceValue, const TRect &ARect, bool EnableFontScale, bool Clip, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawText**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, bool EnableFontScale, bool Clip, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**DrawText**(void * DeviceValue, const TRect &ARect, bool EnableFontScale, bool Clip, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawText**(DeviceValue: THandle; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; EnableFontScale: Boolean; Clip: Boolean; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawText**(DeviceValue: THandle; const ARect: TRect; EnableFontScale: Boolean; Clip: Boolean; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawText**(DeviceValue: Pointer; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; EnableFontScale: Boolean; Clip: Boolean; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**DrawText**(DeviceValue: Pointer; const ARect: TRect; EnableFontScale: Boolean; Clip: Boolean; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
EnableFontScale	フォントのスケーリング［True:する、False:しない］
Clip	クリッピング［True:する、False:しない］
DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ALeft,ATop,ARight,ABottom もしくは ARect で指定した座標を矩形とし、その中にテキストを描画します。
（対象はスクリーン、プリンタ、メモリハンドル）
EnableFontScale が True の場合は、文字列が矩形内に収まりきらない場合にフォントをスケーリングします。ただし、スケーリングを行っても文字列が矩形内に収まりきらない場合は、Clip によって状態が変わります。（例えば、一番小さいフォントサイズを使用しても文字列が矩形内に収まりきらない場合など。）
EnableFontScale が False の場合は、指定されたフォントをそのまま使用し、文字列が境界矩形に収まりきらない場合は収まる最後の単語の切れ目で折り返し、CRLF でも折り返しを行います。（複数行の描画可）
Clip が True の場合は、矩形内に収まりきらない文字列をカットしますが、False の場合は収まりきらない文字列も描画します。矩形内に文字列が収まる場合は、Clip は意味を持ちません。

文字列の描画方向は、文字列の方向と文字の回転に関わらず、与えられた開始位置から右もしくは下方向に描画します。描画するには**CharSet**,**TextColor1**,**TextColor2**,**FontName**,**FontSize**,**Transparent**,**Direction**,**CharAngle**,**CharExtra**,**HCentering**,**VCentering**,**HotkeyPrefix**,**Text**プロパティに値を設定する必要があります。**CharAngle**プロパティは 0,90,180,270 のみ有効です。EnableFontScaleがFalseの場合は**CharAngle**,**Direction**,**VCentering**プロパティの設定は無効です。**HotkeyPrefix**プロパティを 0 に設定するとプリフィックス文字の処理を行いません(&AはAにはならず、&Aと描画されます)。

ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合
　DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合
　DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。
・TRect 型を渡すメソッドが実装されました。

Ellipse（イメージキットコントロール／PrintDraw メソッド）

【機能】
描画先オブジェクトに楕円を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Ellipse**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, bool EnableFontScale, bool Clip, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Ellipse**(NativeUInt DeviceValue, const TRect &ARect, bool EnableFontScale, bool Clip, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Ellipse**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, bool EnableFontScale, bool Clip, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Ellipse**(void * DeviceValue, const TRect &ARect, bool EnableFontScale, bool Clip, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Ellipse**(DeviceValue: THandle; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Ellipse**(DeviceValue: THandle; const ARect: TRect; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Ellipse**(DeviceValue: Pointer; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Ellipse**(DeviceValue: Pointer; const ARect: TRect; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ALeft,ATop,ARight,ABottomもしくはARectで指定した矩形に内接する円を描画します。アウトラインは **PenStyle** プロパティの値で描画され、内部は **BrushStyle** プロパティの値で塗りつぶされます。（対象はスクリーン、プリンタ、メモリハンドル）
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**BrushStyle**,**BrushColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。**BackColor** プロパティは **Transparent** プロパティが False でペンが実線以外、あるいはブラシがハッチパターンの場合に有効です。

ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】

・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。

・TRect 型を渡すメソッドが実装されました。

EnumPaperBins（イメージキットコントロール／PrintDraw メソッド）

【機能】
指定されたプリンタ名からサポートされている用紙トレイの名前、トレイの番号を取得します。

【書式】
(1)C++Builder
[*int* =]*imagekitcontrolname*->**PrintDraw**->**EnumPaperBins**(UnicodeString * BinNames, const int BinNames_Size, Word * BinNumbers, const int BinNumbers_Size)
(2)Delphi
[*Integer* =]*imagekitcontrolname*.**PrintDraw**.**EnumPaperBins**(var BinNames: array of string; var BinNumbers: array of Word)

【引数】

名称	内容
BinNames	用紙トレイの名前を取得する配列
BinNumbers	用紙トレイの番号を取得する配列

※C++Builder の場合、BinNames の要素数-1 を BinNames_Size に与えます。

※C++Builder の場合、BinNumbers の要素数-1 を BinNumbers_Size に与えます。

【戻り値】
取得した項目数を返します（0 は失敗）。

【解説】
配列に必要な要素数は **GetArrayNum** メソッドで取得します。

コード例：
(1)C++Builder
```
int Size;
UnicodeString *BinNames;
Word *BinNumbers;

VImageKit1->PrintDraw->PrinterName = "EPSON LP-8200C";
Size = VImageKit1->PrintDraw->GetArrayNum(vikPrinterPaperBin);
if (Size < 1) return;
BinNames = new UnicodeString[Size];
BinNumbers = new Word[Size];
_try {
  VImageKit1->PrintDraw->EnumPaperBins(BinNames, Size - 1, BinNumbers, Size - 1);
  // 様々な処理
  ・・・・・・・
} _finally {
  delete[] BinNames;
  delete[] BinNumbers;
}
```
(2)Delphi
```
Size: Integer;
BinNames: array of string;
BinNumbers: array of Word;

VImageKit1.PrintDraw.PrinterName := 'EPSON LP-8200C';
Size := VImageKit1.PrintDraw.GetArrayNum(vikPrinterPaperBin);
if Size < 1 then Exit;
SetLength(BinNames, Size);
SetLength(BinNumbers, Size);
```

```
VImageKit1->PrintDraw->EnumPaperBins(BinNames, BinNumbers);
// 様々な処理
・・・・・・・
```

【ImageKit7/8/9/10 ActiveX との違い】

・引数の BinNames が文字列型ではなく文字列型の配列に変更されました。

・C++Builder で使用する場合、引数の BinNames,BinNumbers を指定する配列の要素数を渡す必要があります。

EnumPaperSizes(イメージキットコントロール／PrintDraw メソッド)

【機能】
指定されたプリンタ名からサポートされている用紙の名前、サイズ番号、寸法を取得します。

【書式】
(1)C++Builder
[*int =*]*imagekitcontrolname*->**PrintDraw**->**EnumPaperSizes**(UnicodeString * PaperNames, const int PaperNames_Size, Word * PaperNumbers, const int PaperNumbers_Size, TPoint * PaperSizes, const int PaperSizes_Size)
(2)Delphi
[*Integer =*]*imagekitcontrolname*.**PrintDraw**.**EnumPaperSizes**(var PaperNames: array of string; var PaperNumbers: array of Word; var PaperSizes: array of TPoint)

【引数】

名称	内容
PaperNames	用紙の名前を取得する文字列 ※C++Builder の場合、PaperNames の要素数-1 を PaperNames_Size に与えます。
PaperNumbers	用紙のサイズ番号を取得する配列 ※C++Builder の場合、PaperNumbers の要素数-1 を PaperNumbers_Size に与えます。
PaperSizes	用紙の寸法を取得する配列 ※C++Builder の場合、PaperSizes の要素数-1 を PaperSizes_Size に与えます。

【戻り値】
取得した項目数を返します(0 は失敗)。

【解説】
配列に必要な要素数は **GetArrayNum** メソッドで取得します。
PaperSizes は各用紙サイズの寸法を 0.1mm 単位で取得します。印刷の向きを縦方向として **x** に用紙の幅、**y** に用紙の長さを格納します。

コード例:
(1)C++Builder

```
int Size;
UnicodeString *PaperNames;
Word *PaperNumbers;
TPoint *PaperSizes;

VImageKit1->PrintDraw->PrinterName = "EPSON LP-8200C";
Size = VImageKit1->PrintDraw->GetArrayNum(vikPrinterPaperSize);
if (Size < 1) return;

PaperNames = new UnicodeString[Size];
PaperNumbers = new Word[Size];
PaperSizes = new TPoint[Size];
_try {
   VImageKit1->PrintDraw->EnumPaperSizes(PaperNames, Size - 1, PaperNumbers, Size - 1, PaperSizes, Size - 1);
   // 様々な処理
   ・・・・・・・
} _finally {
   delete[] PaperNames;
   delete[] PaperNumbers;
   delete[] PaperSizes;
}
```

(2)Delphi

```
Size: Integer;
PaperNames: array of string;
PaperNumbers: array of Word;
PaperSizes: array of TPoint;

VImageKit1.PrintDraw.PrinterName := 'EPSON LP-8200C';
Size := VImageKit1.PrintDraw.GetArrayNum(vikPrinterPaperSize);
if Size < 1 then Exit;
SetLength(PaperNames, Size);
SetLength(PaperNumbers, Size);
SetLength(PaperSizes, Size);
VImageKit1->PrintDraw->EnumPaperSizes(PaperNames, PaperNumbers, PaperSizes);
// 様々な処理
・・・・・・・
```

【ImageKit7/8/9/10 ActiveX との違い】

・引数の PaperNames が文字列型ではなく文字列型の配列に変更されました。

・C++Builder で使用する場合、引数の PaperNames,PaperNumbers,PaperSizes を指定する配列の要素数を渡す必要があります。

EnumPorts（イメージキットコントロール／PrintDraw メソッド）

【機能】
ポートを列挙します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**EnumPorts**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**EnumPorts**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
戻り値が True の場合には、**Ports** プロパティに取得したポートの名称が設定されます。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の PortNames が削除されました。
・戻り値が整数型から論理型に変更されました。取得したポートの数は **Ports.Count** で取得できます。

EnumPrinters(イメージキットコントロール／PrintDraw メソッド)

【機能】
インストールされているプリンタを列挙します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**EnumPrinters**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**EnumPrinters**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
戻り値が True の場合には、**Printers** プロパティに取得したプリンタの名称が設定されます。
PrinterIndex プロパティは **Printers** プロパティに設定されたプリンタの中で、通常使うプリンタが何番かを示します。最初のプリンタを示す場合は 0 となります。

デフォルトプリンタに位置付ける例:
(1)C++Builder
```
VImageKit1->PrintDraw->EnumPrinters();
if (VImageKit1->PrintDraw->Printers->Count > 0)
{
   ComboBox1->Items = VImageKit1->PrintDraw->Printers;
   ComboBox1->ItemIndex = VImageKit1->PrintDraw->PrinterIndex;
}
```
(2)Delphi
```
VImageKit1.PrintDraw.EnumPrinters();
if (VImageKit1.PrintDraw.Printers.Count > 0) then
begin
   ComboBox1.Items := VImageKit1.PrintDraw.Printers;
   ComboBox1.ItemIndex := VImageKit1.PrintDraw.PrinterIndex;
end;
```

【ImageKit7/8/9/10 ActiveX との違い】
・引数の DefaultPrinterNo が削除され、**PrinterIndex** プロパティが追加されました。
・戻り値が整数型から論理型に変更されました。取得したプリンタの数は **Printers.Count** で取得できます。

EnumResolutions（イメージキットコントロール／PrintDraw メソッド）

【機能】
指定されたプリンタ名からサポートされている解像度のリストを取得します。

【書式】
(1)C++Builder
[*int =*]*imagekitcontrolname*->**PrintDraw**->**EnumResolutions**(TPoint * Resolutions, const int Resolutions_Size)
(2)Delphi
[*Integer =*]*imagekitcontrolname*.**PrintDraw**.**EnumResolutions**(var Resolutions: array of TPoint)

【引数】

名称	内容
Resolutions	解像度のリストを取得する配列

※C++Builder の場合、Resolutions の要素数-1 を Resolutions_Size に与えます。

【戻り値】
取得した項目数を返します（0 は失敗）。

【解説】
配列に必要な要素数は **GetArrayNum** メソッドで取得します。
Resolutions には 1 インチ当たりのドット数（dpi）単位で水平解像度と垂直解像度を示す値が格納されます。**x** が水平解像度、**y** が垂直解像度になります。

コード例：
(1)C++Builder

```
int Size;
TPoint *Resolutions;

VImageKit1->PrintDraw->PrinterName = "EPSON LP-8200C";
Size = VImageKit1->PrintDraw->GetArrayNum(vikPrinterResolution);
if (Size < 1) return;
Resolutions = new TPoint[Size];
_try {
   VImageKit1->PrintDraw->EnumResolutions(Resolutions, Size - 1);
   // 様々な処理
   ・・・・・・・
} _finally {
   delete[] Resolutions;
}
```

(2)Delphi

```
Size: Integer;
Resolutions: array of TPoint;

VImageKit1.PrintDraw.PrinterName := 'EPSON LP-8200C';
Size := VImageKit1.PrintDraw.GetArrayNum(vikPrinterResolution);
if Size < 1 then Exit;
SetLength(Resolutions, Size);
VImageKit1.PrintDraw.EnumResolutions(Resolutions);
// 様々な処理
・・・・・・・
```

【ImageKit7/8/9/10 ActiveX との違い】
・引数の XResolutions,YResolutions が TPoint 型に変更されました。
・C++Builder で使用する場合、引数の Resolutions を指定する配列の要素数を渡す必要があります。

FillRect(イメージキットコントロール／PrintDraw メソッド)

【機能】
描画先オブジェクトに対して指定した矩形を塗りつぶします。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**FillRect**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**FillRect**(NativeUInt DeviceValue, const TRect &ARect, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**FillRect**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**FillRect**(void * DeviceValue, const TRect &ARect, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**FillRect**(DeviceValue: THandle; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**FillRect**(DeviceValue: THandle; const ARect: TRect; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**FillRect**(DeviceValue: Pointer; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**FillRect**(DeviceValue: Pointer; const ARect: TRect; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル(DeviceMode による)
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
DeviceMode	描画対象(vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル)

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ALeft,ATop,ARight,ABottom もしくは ARect で指定した矩形を **BrushStyle** プロパティの値で塗りつぶします。ただし、矩形の左と上の端は領域に含まれますが、右と下の端は含まれません。(対象はスクリーン、プリンタ、メモリハンドル)
描画するには **BrushStyle**,**BrushColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。**BackColor** プロパティは **Transparent** プロパティが False でハッチパターンブラシの場合に有効です。**Transparent** プロパティが True でハッチパターンブラシを選択した場合、線と線の間は透過されません(既定の色で塗りつぶされます)。線と線の間を透過する場合は **Rectangle** メソッドなどをご使用ください。

ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。
・TRect 型を渡すメソッドが実装されました。

FrameRect（イメージキットコントロール／PrintDraw メソッド）

【機能】
描画先オブジェクトに対して指定した矩形の境界を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**FrameRect**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**FrameRect**(NativeUInt DeviceValue, const TRect &ARect; TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**FrameRect**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**FrameRect**(void * DeviceValue, const TRect &ARect; TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**FrameRect**(DeviceValue: THandle; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**FrameRect**(DeviceValue: THandle; const ARect: TRect; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**FrameRect**(DeviceValue: Pointer; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**FrameRect**(DeviceValue: Pointer; const ARect: TRect; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ALeft,ATop,ARight,ABottomもしくはARectで指定した矩形を**BrushStyle**プロパティの値で描画します。境界の内部は塗りつぶされません。（対象はスクリーン、プリンタ、メモリハンドル）
描画するには**BrushStyle**,**BrushColor**,**Transparent**,**BackColor**プロパティに値を設定する必要があります。**BackColor**プロパティは**Transparent**プロパティがFalseでハッチパターンブラシの場合に有効です。**Transparent**プロパティがTrueでハッチパターンブラシを選択した場合、線と線の間は透過されません（既定の色で塗りつぶされます）。線と線の間を透過する場合は**Rectangle**メソッドなどをご使用ください。

ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。
・TRect 型を渡すメソッドが実装されました。

GetArrayNum(イメージキットコントロール／PrintDraw メソッド)

【機能】
指定されたプリンタ名からサポートされている項目を取得する際に必要な配列の要素数を取得します。

【書式】
(1)C++Builder　[*int =*]*imagekitcontrolname*->**PrintDraw->GetArrayNum**(TVIkPrinterCapability CapNo)
(2)Delphi　[*Integer =*]*imagekitcontrolname*.**PrintDraw.GetArrayNum**(CapNo: TVIkPrinterCapability)

【TVIkPrinterCapability 型】
ユニット
ImageKit
type
TVIkPrinterCapability = (vikPrinterPaperSize, vikPrinterPaperBin, vikPrinterResolution);

【引数】

名称	内容
CapNo	取得する項目の種類 (vikPrinterPaperSize:用紙サイズ、vikPrinterPaperBin:用紙トレイ、vikPrinterResolution:解像度)

【戻り値】
取得する配列に必要な要素数を返します(0 は失敗)。

【解説】
EnumPaperSizes,**EnumPaperBins**,**EnumResolutions** メソッドで引数として渡す配列の要素数を取得します。

コード例:
(1)C++Builder
```
int Size;
TPoint *Resolutions;

VImageKit1->PrintDraw->PrinterName = "EPSON LP-8200C";
Size = VImageKit1->PrintDraw->GetArrayNum(vikPrinterResolution);
if (Size < 1) return;
Resolutions = new TPoint[Size];
_try {
  VImageKit1->PrintDraw->EnumResolutions(Resolutions, Size - 1);
  // 様々な処理
  ・・・・・・・
} _finally {
  delete[] Resolutions;
}
```
(2)Delphi
```
Size: Integer;
Resolutions: array of TPoint;

VImageKit1.PrintDraw.PrinterName := 'EPSON LP-8200C';
Size := VImageKit1.PrintDraw.GetArrayNum(vikPrinterResolution);
if Size < 1 then Exit;
SetLength(Resolutions, Size);
VImageKit1.PrintDraw.EnumResolutions(Resolutions);
// 様々な処理
・・・・・・・
```

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikPrinterPaperSize, ikPrinterPaperBin, ikPrinterResolution)。

GetDefaultPrinter(イメージキットコントロール／PrintDraw メソッド)

【機能】
通常使うプリンタ(デフォルトプリンタ)を取得します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**GetDefaultPrinter**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**GetDefaultPrinter**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Windows のプリンタフォルダに登録されている通常使うプリンタを取得します。成功すると、**PrinerName** プロパティに通常使うプリンタの名称が設定されます。**EnumPrinters** メソッドでも通常使うプリンタを取得できます。

GetDevModeHandle(イメージキットコントロール／PrintDraw メソッド)

【機能】
指定したプリンタの DEVMODE 構造体へのポインタのハンドルとDEVNAMES 構造体へのポインタのハンドルを取得します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**GetDevModeHandle**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**GetDevModeHandle**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
PrinterName プロパティ(プリンタ名)もしくは **PrintFileName** プロパティ(**SaveDevModeHandle** もしくは **SavePrinterInfo** メソッドで保存したプリンタ設定ファイル名)を設定して実行してください。両方のプロパティを設定した場合は **PrinterName** プロパティが有効になります。

プリンタ名から取得する場合の例

```
VImageKit1.PrintDraw.PrinterName := 'EPSON LP-8200C';
VImageKit1.PrintDraw.PrintFileName := '';
VImageKit1.PrintDraw.GetDevModeHandle;
//様々な処理
VImageKit1.PrintDraw.ReleaseDevModeHandle;
```

プリンタ設定ファイルから取得する場合の例

```
VImageKit1.PrintDraw.PrinterName := '';
VImageKit1.PrintDraw.PrintFileName := 'IkPrn.lk';
VImageKit1.PrintDraw.GetDevModeHandle;
//様々な処理
VImageKit1.PrintDraw.ReleaseDevModeHandle;
```

成功すると、**DevMode** プロパティに DEVMODE 構造体へのポインタのハンドルが設定され、**DevNames** プロパティに DEVNAMSE 構造体へのポインタのハンドルが設定されます。
DevMode プロパティと **DevNames** プロパティは **PrintCreateDC** メソッドなどで使用します。DEVMODE 構造体や DEVNAMES 構造体については WindowsAPI 関連の書籍などをご覧ください。
なお、DEVMODE 構造体のハンドルと DEVNAMES 構造体のハンドルは **ReleaseDevModeHandle** メソッドで解放します。

【ImageKit7/8/9 ActiveX/VCL との違い】
当メソッドが成功すると **DevMode** プロパティに加え **DevNames** プロパティの値が更新されます。

GetDevModeInfo（イメージキットコントロール／PrintDraw メソッド）

【機能】
DEVMODE 構造体のハンドルから印刷情報を取得します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw->GetDevModeInfo**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw.GetDevModeInfo**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
GetDevModeInfo メソッドを実行するには、**GetDevModeHandle** メソッドで **DevMode** プロパティを取得する必要があります。
成功した場合、**Collate**,**ColorMode**,**Copies**,**CustomPaperHeight**,**CustomPaperWidth**,**Duplex**,**Orientation**,**PaperBin**,**PaperSize**,**XResolution**,**YResolution**,**Zoom** プロパティに値が設定されます。
このメソッドを実行しなくても **GetDevModeHandle** メソッドを実行して成功すると各種情報は自動的に設定されます。

GetDevNamesInfo（イメージキットコントロール／PrintDraw メソッド）

【機能】

DEVNAMES 構造体のハンドルからプリンタ名とポート名を取得します。

【書式】

(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw->GetDevNamesInfo**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw.GetDevNamesInfo**

【引数】

ありません。

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

GetDevNamesInfo メソッドを実行するには、**GetDevModeHandle** メソッドで **DevNames** プロパティを取得する必要があります。
成功した場合、**PrinterName**,**PortName** プロパティに値が設定されます。
このメソッドを実行しなくても **GetDevModeHandle** メソッドを実行して成功すると文字列情報は自動的に設定されます。

GetImageFromHdc(イメージキットコントロール／PrintDraw メソッド)

【機能】
スクリーンのデバイスコンテキストに描画されたイメージを取得します。

【書式】
(1)C++Builder
[*NativeUInt =*]*imagekitcontrolname*->**PrintDraw**->**GetImageFromHdc**(HDC DC, int AWidth, int AHeight, short BitCount)
(2)Delphi
[*THandle =*]*imagekitcontrolname*.**PrintDraw**.**GetImageFromHdc**(DC: HDC; AWidth, AHeight: Integer; BitCount: Smallint)

【引数】

名称	内容
DC	スクリーンのデバイスコンテキスト
AWidth	取得するイメージの幅(ピクセル)
AHeight	取得するイメージの高さ(ピクセル)
BitCount	作成したいイメージのビット数

【戻り値】
ラスタイメージのメモリハンドル(実行に失敗した場合は 0 が返されます)。

【解説】
スクリーンのデバイスコンテキストに描画されたイメージを取得し、メモリハンドルとして返します。
実際のイメージの幅と高さが取得できないコンポーネントからのイメージの取得は、上手くいかない場合があります。

GetPaperSize(イメージキットコントロール／PrintDraw メソッド)

【機能】
プリンタのデバイスコンテキストから印刷有効領域と用紙サイズを取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**GetPaperSize**(HDC DC, int &ALeft, int &ATop, int &ARight, int &ABottom, int &AWidth, int &AHeight, TVIkDeviceUnitMode UnitMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**GetPaperSize**(HDC DC, TRect &ARect, int &AWidth, int &AHeight, TVIkDeviceUnitMode UnitMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**GetPaperSize**(DC: HDC; var ALeft: Integer; var ATop: Integer; var ARight: Integer; var ABottom: Integer; var AWidth: Integer; var AHeight: Integer; UnitMode: TVIkDeviceUnitMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**GetPaperSize**(DC: HDC; var ARect: TRect; var AWidth: Integer; var AHeight: Integer; UnitMode: TVIkDeviceUnitMode)

【TVIkDeviceUnitMode 型】
ユニット
ImageKit
type
TVIkDeviceUnitMode = (vikPrinterPixel, vikPrinterMM);

【引数】

名称	内容
DC	プリンタのデバイスコンテキスト
ALeft,ATop	印刷有効領域の左上の位置を取得する変数
ARight,ABottom	印刷有効領域の右下の位置を取得する変数
ARect	印刷有効領域の矩形を取得する変数
AWidth	用紙の幅を取得する変数
AHeight	用紙の高さを取得する変数
UnitMode	取得する単位(vikPrinterPixel:ピクセル、vikPrinterMM:0.1mm)

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
プリンタのデバイスコンテキストから印刷有効領域と用紙サイズを取得します。

ALeft,ATop,ARight,ABottom(もしくはARect),AWidth,AHeightをピクセル単位として扱う場合
UnitMode が vikPrinterPixel
ALeft,ATop,ARight,ABottom(もしくはARect),AWidth,AHeightを 0.1mm単位として扱う場合
UnitMode が vikPrinterMM

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikPrinterPixel, ikPrinterMM)。
・TRect 型を渡すメソッドが実装されました。

GetPixel(イメージキットコントロール／PrintDraw メソッド)

【機能】
描画先オブジェクトの指定したピクセルから RGB 値を取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**GetPixel**(NativeUInt DeviceValue, int X, int Y, Byte &Red, Byte &Green, Byte &Blue, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**GetPixel**(void * DeviceValue, int X, int Y, Byte &Red, Byte &Green, Byte &Blue, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**GetPixel**(DeviceValue: THandle; X: Integer; Y: Integer; var Red: Byte; var Green: Byte; var Blue: Byte; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**GetPixel**(DeviceValue: Pointer; X: Integer; Y: Integer; var Red: Byte; var Green: Byte; var Blue: Byte; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル(DeviceMode による)
X,Y	ピクセルの座標
Red	RGB の赤のパレット
Green	RGB の緑のパレット
Blue	RGB の青のパレット
DeviceMode	描画対象(vikScreen:スクリーン、vikMemoryHandle:メモリハンドル)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
x,y で指定したピクセルから RGB 値を取得します。(対象はスクリーン、メモリハンドル)
DeviceMode が vikScreen もしくは vikMemoryHandle の場合に有効になります。x,y の値はピクセル単位で設定してください。

【ImageKit7/8/9/10 ActiveX との違い】
・引数の Red, Green, Blue が Byte 型に変更されました。
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikMemoryHandle)。

GetPrinterPort（イメージキットコントロール／PrintDraw メソッド）

【機能】
プリンタのポートを取得します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**GetPrinterPort**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**GetPrinterPort**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
メソッドを実行する前に **PrinterName** プロパティにプリンタ名を設定する必要があります。
成功した場合、**PortName** プロパティにポート名が設定されます。
プリンタが複数のポートに接続されている場合、各ポート名はカンマで区切られます。
例：″LPT1:,LPT2:″

GetTextExtent(イメージキットコントロール／PrintDraw メソッド)

【機能】
テキストの情報から文字列の幅と高さを取得します。

【書式】
(1)C++Builder
[*TSize =*]*imagekitcontrolname*->**PrintDraw**->**GetTextExtent**(HDC DC, TVIkDeviceUnitMode UnitMode)
(2)Delphi
[*TSize =*]*imagekitcontrolname*.**PrintDraw**.**GetTextExtent**(DC: HDC; UnitMode: TVIkDeviceUnitMode)

【TVIkDeviceUnitMode 型】
ユニット
ImageKit
type
TVIkDeviceUnitMode = (vikPrinterPixel, vikPrinterMM);

【引数】

名称	内容
DC	デバイスコンテキスト
UnitMode	取得する単位(vikPrinterPixel:ピクセル、vikPrinterMM:0.1mm)

【戻り値】
文字列の幅と高さ(失敗の場合は両方とも 0)
TSize.cx に文字列の幅が、TSize.cy に高さが設定されます。

【解説】
テキストの情報から文字列の幅と高さを取得します。(単一行のみ対応)
CharSet,**FontName**,**FontSize**,**Direction**,**CharAngle**,**RotateString** プロパティに値を設定する必要があります。
CharExtra プロパティに 1 以上の値が設定されていた場合は、文字間を考慮した値になります。
RotateString プロパティが False の場合は文字列の方向と文字の回転に関わらず、文字列を描画する横方向が幅で縦方向が高さとなります。**RotateString** プロパティが True の場合は文字列を描画する方向が幅となり、垂直方向が高さとなります。
RotateString プロパティが False の場合、**CharAngle** プロパティは 0,90,180,270 以外は無効です。

文字列の幅と高さをピクセル単位として返す場合
UnitMode が vikPrinterPixel
文字列の幅と高さを 0.1mm単位として返す場合
UnitMode が vikPrinterMM

【ImageKit7/8/9/10 ActiveX との違い】
・引数の Width,Heigh が削除され、戻り値に変更されました。
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikPrinterPixel, ikPrinterMM)。

ImageOut（イメージキットコントロール／PrintDraw メソッド）

【機能】
デバイスコンテキストにイメージを描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**ImageOut**(HDC DC, NativeUInt AHandle, int ALeft, int ATop, int ARight, int ABottom, bool Aspect, bool DXFBlack, TVIkDeviceUnitMode UnitMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**ImageOut**(HDC DC, NativeUInt AHandle, const TRect &ARect, bool Aspect, bool DXFBlack, TVIkDeviceUnitMode UnitMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**ImageOut**(DC: HDC; AHandle: THandle; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; Aspect: Boolean; DXFBlack: Boolean; UnitMode: TVIkDeviceUnitMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**ImageOut**(DC: HDC; AHandle: THandle; const ARect: TRect; Aspect: Boolean; DXFBlack: Boolean; UnitMode: TVIkDeviceUnitMode)

【TVIkDeviceUnitMode 型】
ユニット
ImageKit
type
TVIkDeviceUnitMode = (vikPrinterPixel, vikPrinterMM);

【引数】

名称	内容
DC	デバイスコンテキスト
AHandle	ラスタイメージもしくはベクトルイメージのメモリハンドル
ALeft,ATop	イメージを描画する矩形の左上の x,y 座標
ARight,ABottom	イメージを描画する矩形の右下の x,y 座標
ARect	イメージを描画する矩形の x,y 座標
Aspect	縦横比の設定［True:する、False:しない］
DXFBlack	描画イメージが DXF の場合、白を黒に置き換えて描画［ True:する　False:しない ］
UnitMode	描画時の単位（vikPrinterPixel:ピクセル、vikPrinterMM:0.1mm）

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。
（印刷処理中にダイアログボックスを表示してキャンセルした場合は、False を返します。）

【解説】
イメージが ALeft,ATop,ARight,ABottom もしくは ARect で指定された範囲に収まりきらない場合には、イメージをスケーリングします。
ALeft,ATop,ARight,ABottom もしくは ARect にイメージをそのまま描画する場合は Aspect に False を、イメージの縦横比と描画領域の縦横比を考慮する場合は Aspect に True を設定してください。
（描画対象はスクリーン、プリンタ）

ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合
UnitMode が vikPrinterPixel
ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合
UnitMode が vikPrinterMM

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加され、名称が変更されました（ActiveX は ikScreen, ikPrinter, ikMemoryHandle）。
・TRect 型を渡すメソッドが実装されました。

ImageOutToHwnd(イメージキットコントロール／PrintDraw メソッド)

【機能】
ウィンドウハンドルに対してイメージを描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**ImageOutToHwnd**(HWND Wnd, NativeUInt AHandle, TColor AColor, bool Aspect, bool DXFBlack)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**ImageOutToHwnd**(Wnd: HWND; AHandle: THandle; AColor: TColor; Aspect: Boolean; DXFBlack: Boolean)

【引数】

名称	内容
Wnd	ウィンドウハンドル
AHandle	ラスタイメージもしくはベクトルイメージのメモリハンドル
AColor	ウィンドウの背景色を設定します。
Aspect	縦横比の設定［True:する、False:しない］
DXFBlack	描画イメージが DXF の場合、白を黒に置き換えて描画［ True:する　False:しない ］

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
イメージがウィンドウの描画領域に収まりきらない場合には、イメージをスケーリングします。
ウィンドウの描画領域にイメージをそのまま描画する場合は Aspect に False を、イメージの縦横比と描画領域の縦横比を考慮する場合は Aspect に True を設定してください。
(描画対象はスクリーン)
AColor には色定数(clRed など)や **RGB**(Red,Green,Blue)として求めた値などを設定してください。

Line（イメージキットコントロール／PrintDraw メソッド）

【機能】
描画先オブジェクトに直線を描画します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**PrintDraw**->**Line**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, TVIkOutPutDeviceMode DeviceMode)
[*bool* =]*imagekitcontrolname*->**PrintDraw**->**Line**(NativeUInt DeviceValue, const TRect &ARect, TVIkOutPutDeviceMode DeviceMode)
[*bool* =]*imagekitcontrolname*->**PrintDraw**->**Line**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, TVIkOutPutDeviceMode DeviceMode)
[*bool* =]*imagekitcontrolname*->**PrintDraw**->**Line**(void * DeviceValue, const TRect &ARect, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**PrintDraw**.**Line**(DeviceValue: THandle; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean* =]*imagekitcontrolname*.**PrintDraw**.**Line**(DeviceValue: THandle; const ARect: TRect; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean* =]*imagekitcontrolname*.**PrintDraw**.**Line**(DeviceValue: Pointer; ALeft: Integer; ATop: Integer; ARight: Integer; ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean* =]*imagekitcontrolname*.**PrintDraw**.**Line**(DeviceValue: Pointer; const ARect: TRect; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
ALeft,ATop	始点の x,y 座標
ARight,ABottom	終点の x,y 座標
ARect	始点終点を表す矩形
DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ALeft,ATop で指定された座標から、ARight,ABottom で指定された座標まで直線を描画します（ARect を使用する場合は Left と Top に始点を、Right と Bottom に終点を設定します）。
ただし、ARight,ABottom の点自体は線に含まれず描画されません。（対象はスクリーン、プリンタ、メモリハンドル）
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。
BackColor プロパティは **Transparent** プロパティが False でペンが実線以外の場合に有効です。

ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。
・TRect 型を渡すメソッドが実装されました。

MeasureString(イメージキットコントロール／PrintDraw メソッド)

【機能】
GDI+の機能を利用してテキストの情報から文字列の幅と高さを取得します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**MeasureString**(HDC, int AreaWidth, int AreaHeight, float &AWidth, float &AHeight, int &charactersFitted, int &linesFilled, int FormatFlags, TVIkDeviceUnitMode UnitMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**MeasureString**(DC: HDC; AreaWidth: Integer; AreaHeight: Integer; var AWidth: Single; var AHeight: Single; var charactersFitted: Integer; var linesFilled: Integer; FormatFlags: Integer; UnitMode: TVIkDeviceUnitMode)

【TVIkDeviceUnitMode 型】
ユニット
ImageKit
type
TVIkDeviceUnitMode = (vikPrinterPixel, vikPrinterMM);

【引数】

名称	内容
DC	デバイスコンテキスト
AreaWidth	矩形領域の最大幅(0～)
AreaHeight	矩形領域の最大高さ(0～)
AWidth	文字列の幅を取得する変数
AHeight	文字列の高さを取得する変数
charactersFitted	取得する文字列の文字数
linesFilled	取得する文字列の行数
FormatFlags	文字列の書式情報(デフォルトは 0)
UnitMode	計測時の単位(vikPrinterPixel:ピクセル、vikPrinterMM:0.1mm)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
文字列の幅と高さを取得するには **FontName**,**FontSize**,**HCentering**,**VCentering**,**HotkeyPrefix**,**Text** プロパティに値を設定する必要があります。AreaWidth と AreaHeight のどちらかに 0 を設定した場合は、**Text** プロパティに与えられた文字列の幅と高さを取得しますが、AreaWidth と AreaHeight に 1 以上を与えた場合は、その矩形領域に含まれる文字列の幅と高さを取得します。charactersFitted と linesFilled についても同様な扱いとなります。

AreaWidth,AreaHeight,AWidth,AHeightをピクセル単位として返す場合
UnitMode が vikPrinterPixel
AreaWidth,AreaHeight,AWidth,AHeightを 0.1mm単位として返す場合
UnitMode が vikPrinterMM

文字列の書式情報を示します。FormatFlags に設定する値は次の値の論理和となります。

値	説明
0x00000001	テキストの方向を右から左に指定します。
0x00000002	テキストが縦書きになるよう指定します。
0x00000004	グリフが外接する四角形にかからないよう指定します。既定では、端に表示する必要がある場合、一部のグリフが若干四角形にかかるよう設定されます。
0x00000020	左から右を指示するマークなどの制御文字をグリフで表現します。
0x00000400	フォールバックを無効にして、要求されたフォントでサポートされていない文字のフォントを切り替えます。

	欠落文字は、グリフの欠落したフォント(通常は空白の正方形)で表示されます。
0x00000800	既定では、各行末の空白が除外されます。各行末の空白を計測に含める場合はこの値を設定します。
0x00001000	四角形内の書式指定時に、行間のテキストのラップを無効にします。この値は、四角形ではなく点が渡された場合、または長さゼロの行の四角形が指定された場合に暗黙的に指定されます。
0x00002000	書式指定用の四角形には、完全な直線だけがレイアウトされます。既定では、クリッピングの結果、テキストの末尾が表示された状態、または行が表示されなくなった状態のうち、いずれか早い方の状態になるまでレイアウトが継続します。既定の設定では、行高さの整数倍でない書式指定用四角形を用いた場合は、なるまでレイアウトが継続します。既定の設定では、行高さの整数倍でない書式指定用四角形を用いた場合は、最後の行の一部が隠れることがあります。必ず行全体が表示されるようにするには、この値を指定した上で、少なくとも 1 つの行と高さが同じの書式指定用の四角形をご使用ください。
0x00004000	論理グリフの突出部と書式指定用の四角形からはみ出すラップされていないテキストを表示できます。既定では、書式指定用の四角形からはみ出たテキストとグリフ部はすべてクリップされます。

定数(vikDirectionRightToLeft = 0x00000001, vikDirectionVertical = 0x00000002, vikNoFitBlackBox = 0x00000004, vikDisplayFormatControl = 0x00000020, vikNoFontFallback = 0x00000400, ikMeasureTrailingSpaces = 0x00000800, vikNoWrap = 0x00001000, vikLineLimit = 0x00002000, vikNoClip = 0x00004000)を使用することも可能です。

※16 進表記のため、Delphi は 0x を$に置き換えてください。

コード例:

(1)C++Builder

```
float Str_Width, Str_Height;
int charactersFitted, linesFilled, ARight, ABottom;

VImageKit1->PrintDraw->ClearProperty();
VImageKit1->PrintDraw->FontSize = 20;
VImageKit1->PrintDraw->FontName = "MS ゴシック";
VImageKit1->PrintDraw->HotkeyPrefix = 0;
VImageKit1->PrintDraw->Text = "テキスト";
VImageKit1->PrintDraw->MeasureString(DC, 0, 0, Str_Width, Str_Height, charactersFitted, linesFilled, 0,
vikPrinterPixel);

ARight = 10 + (int)Str_Width - 1;
ABottom = 10 + (int)Str_Height - 1;
VImageKit1->PrintDraw->CharAngle = 0;
VImageKit1->PrintDraw->TextColor1 = clRed;
VImageKit1->PrintDraw->Alpha1 = 255;
VImageKit1->PrintDraw->TextColor2 = clWhite;
VImageKit1->PrintDraw->Alpha2 = 255;
//グラデーションブラシを使用
VImageKit1->PrintDraw->DrawString(hDC, 10, 10, ARight, ABottom, 4, 0, 0, 0, 0,
TVIkOutPutDeviceMode(vikPrinterPixel));
```

(2)Delphi

```
Str_Width, Str_Height: Single;
CharactersFitted, linesFilled, TxRight, TxBottom: Integer;

VImageKit1.PrintDraw.ClearProperty();
VImageKit1.PrintDraw.FontSize := 20;
VImageKit1.PrintDraw.FontName := 'MS ゴシック';
VImageKit1.PrintDraw.HotkeyPrefix := 0;
VImageKit1.PrintDraw.Text := 'テキスト';
VImageKit1.PrintDraw.MeasureString(DC, 0, 0, Str_Width, Str_Height, charactersFitted, linesFilled, 0,
vikPrinterPixel);

TxRight := 10 + Trunc(Str_Width) - 1;
TxBottom := 10 + Trunc(Str_Height) - 1;
VImageKit1.PrintDraw.CharAngle := 0;
VImageKit1.PrintDraw.TextColor1 := clRed;
VImageKit1.PrintDraw.Alpha1 := 255;
```

```
VImageKit1.PrintDraw.TextColor2 := clWhite;
VImageKit1.PrintDraw.Alpha2 := 255;
//グラデーションブラシを使用
VImageKit1.PrintDraw.DrawString(DC, 10, 10, TxRight, TxBottom, 4, 0, 0, 0, 0,
TVIkOutPutDeviceMode(vikPrinterPixel));
```

【ImageKit7/8/9/10 ActiveX との違い】

・列挙型の識別子の先頭に v が付加されました（ActiveX は ikPrinterPixel, ikPrinterMM）。

・定数を使用する場合、識別子の先頭に v が付加されました（ActiveX は ikDirectionRightToLeft, ikDirectionVertical, ikNoFitBlackBox, ikDisplayFormatControl, ikNoFontFallback, ikMeasureTrailingSpaces, ikNoWrap, ikLineLimit, ikNoClip）。

Paint（イメージキットコントロール／PrintDraw メソッド）

【機能】
描画先オブジェクトの指定した点を基準に別の色で塗りつぶします。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Paint**(NativeUInt DeviceValue, int X, int Y, TColor Color1, TColor Color2, TVIkPaintMode PaintMode, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Paint**(void * DeviceValue, int X, int Y, TColor Color1, TColor Color2, TVIkPaintMode PaintMode, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Paint**(DeviceValue: THandle; X: Integer; Y: Integer; Color1: TColor; Color2: TColor; PaintMode: TVIkPaintMode; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Paint**(DeviceValue: Pointer; X: Integer; Y: Integer; Color1: TColor; Color2: TColor; PaintMode: TVIkPaintMode; DeviceMode: TVIkOutPutDeviceMode)

【TVIkPaintMode 型】
ユニット
IkInit
type
TVIkPaintMode = (vikBorder, vikSurface);

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
X,Y	開始座標
Color1	塗りつぶされた領域色、もしくは囲まれた領域の境界色
Color2	塗りつぶす色
PaintMode	描画方法
DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
PaintModeがvikBorderの場合
X,Y を開始点として Color1 を境界色として囲まれた領域の内部（外部）を Color2 で塗りつぶします。
PaintModeがvikSurfaceの場合
X,Y を開始点として Color1 で塗りつぶされた領域を Color2 で塗りつぶします。

X,Y から Color1 に突き当たるまで全ての方向に塗りつぶしが行われます。
（対象はスクリーン、プリンタ、メモリハンドル）
Color1,Color2 には色定数(clRed など)や **RGB**(Red,Green,Blue)として求めた値などを設定してください。

X,Yをピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
X,Yを 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】

列挙型の識別子の先頭に v が付加されました(ActiveX は PaintMode が vikBorder, vikSurface、DeviceMode が ikScreen, ikPrinter, ikMemoryHandle)。

Pie(イメージキットコントロール／PrintDraw メソッド)

【機能】
描画先オブジェクトに扇形を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Pie**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Pie**(NativeUInt DeviceValue, const TRect &ARect, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Pie**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Pie**(void * DeviceValue, const TRect &ARect, int X1, int Y1, int X2, int Y2, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Pie**(DeviceValue: THandle; ALeft, ATop, ARight, ABottom, X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Pie**(DeviceValue: THandle; const ARect: TRect; X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Pie**(DeviceValue: Pointer; ALeft, ATop, ARight, ABottom, X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Pie**(DeviceValue: Pointer; const ARect: TRect; X1, Y1, X2, Y2: Integer; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル(DeviceMode による)
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
X1,Y1	楕円弧の始点の x,y 座標
X2,Y2	楕円弧の終点の x,y 座標
DeviceMode	描画対象(vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル)

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
楕円の一部分を描画し、2 つの端点と楕円の中心を接続して扇形を作成します。アウトラインは **PenStyle** プロパティの値で描画され、内部は **BrushStyle** プロパティの値で塗りつぶされます。端点が円弧の上にある必要はなく、中心から指定の端点まで引いた直線との交点が計算され、その交点が用いられます。
(描画対象はスクリーン、プリンタ、メモリハンドル)
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**BrushStyle**,**BrushColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。**BackColor** プロパティは **Transparent** プロパティが False でペンが実線以外、あるいはブラシがハッチパターンの場合に有効です。

ALeft,ATop,ARight,ABottom(もしくはARect),X1,Y1,X2,Y2 をピクセル単位として扱う場合
　DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect),X1,Y1,X2,Y2 を 0.1mm単位として扱う場合
　DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。
・TRect 型を渡すメソッドが実装されました。

PolyBezier（イメージキットコントロール／PrintDraw メソッド）

【機能】
描画先オブジェクトにベジェ曲線を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**PolyBezier**(NativeUInt DeviceValue, const TPoint * Points, const int Points_Size, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**PolyBezier**(void * DeviceValue, const TPoint * Points, const int Points_Size, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**PolyBezier**(DeviceValue: THandle; const Points: array of TPoint; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**PolyBezier**(DeviceValue: Pointer; const Points: array of TPoint; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
Points	多角形の x,y 座標を指定する配列

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ベジェ曲線は始点、2 つの制御点、終点の 4 つの点から構成されます。最初の曲線は 4 つの点で描画されますが、それ以降の曲線は 3 つの点で描画されます。（描画対象はスクリーン、プリンタ、メモリハンドル）
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。
BackColor プロパティは **Transparent** プロパティが False でペンが実線以外の場合に有効です。
Points は「3×曲線の数＋最初の始点」の数だけ必要です。

Pointsの配列の要素の値をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
Pointsの配列の要素の値を 0.1mm単位として扱う場合
DeviceMode が vikPrinter

(1)C++Builder

```
TPoint Points[4];

Points[0].x = 10; Points[0].y = 100;
Points[1].x = 50; Points[1].y = 50;
Points[2].x = 100; Points[2].y = 150;
Points[3].x = 150; Points[3].y = 100;
VImageKit1->PrintDraw->ClearProperty();
VImageKit1->PrintDraw->PenColor = clGreen;
VImageKit1->PrintDraw->PenStyle = vikPenSolid;
```

```
VImageKit1->PrintDraw->PenWidth = 15;
VImageKit1->PrintDraw->PolyBezier(DC, Points, 3, vikScreen);
```

(2)Delphi

```
Points: array[0..3] of TPoint;

Points[0].x := 10; Points[0].y := 100;
Points[1].x := 50; Points[1].y := 50;
Points[2].x := 100; Points[2].y := 150;
Points[3].x := 150; Points[3].y := 100;
VImageKit1.PrintDraw.ClearProperty;
VImageKit1.PrintDraw.PenColor := clGreen;
VImageKit1.PrintDraw.PenStyle := vikPenSolid;
VImageKit1.PrintDraw.PenWidth := 15;
VImageKit1.PrintDraw.PolyBezier(DC, Points, vikScreen);
```

【ImageKit7/8/9/10 ActiveX との違い】

・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。

Polygon(イメージキットコントロール／PrintDraw メソッド)

【機能】
描画先オブジェクトに多角形を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Polygon**(NativeUInt DeviceValue, const TPoint * Points, const int Points_Size, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Polygon**(void * DeviceValue, const TPoint * Points, const int Points_Size, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Polygon**(DeviceValue: THandle; const Points: array of TPoint; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Polygon**(DeviceValue: Pointer; const Points: array of TPoint; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル(DeviceMode による)
Points	多角形の x,y 座標を指定する配列

※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。

DeviceMode	描画対象(vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
指定の点を基にして一連の直線を描画し、最後の点から最初の点まで直線を引くことで図形をクローズします。アウトラインは **PenStyle** プロパティの値で描画され、内部は **BrushStyle** プロパティの値で塗りつぶされます。(描画対象はスクリーン、プリンタ、メモリハンドル)
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**BrushStyle**,**BrushColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。**BackColor** プロパティは **Transparent** プロパティが False でペンが実線以外、あるいはブラシがハッチパターンの場合に有効です。

Pointsの配列の要素の値をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
Pointsの配列の要素の値を 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。

Polyline（イメージキットコントロール／PrintDraw メソッド）

【機能】
描画先オブジェクトに連続線を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Polyline**(NativeUInt DeviceValue, const TPoint * Points, const int Points_Size, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Polyline**(void * DeviceValue, const TPoint * Points, const int Points_Size, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Polyline**(DeviceValue: THandle; const Points: array of TPoint; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Polyline**(DeviceValue: Pointer; const Points: array of TPoint; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
Points	連続線の x,y 座標を指定する配列
※C++Builder の場合、Points の要素数-1 を Points_Size に与えます。	
DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
指定の点で一連の直線を **PenStyle** プロパティの値で描画します。（描画対象はスクリーン、プリンタ、メモリハンドル）
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。
BackColor プロパティは **Transparent** プロパティが False でペンが実線以外の場合に有効です。

x,yの配列の要素の値をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
x,yの配列の要素の値を 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・引数の x,y が TPoint 型に変更されました。
・Delphi で使用する場合、x,y を指定する配列の要素数を引数として渡す必要がなくなりました。
・列挙型の識別子の先頭に v が付加されました（ActiveX は ikScreen, ikPrinter, ikMemoryHandle）。

PreviewInit（イメージキットコントロール／PrintDraw メソッド）

【機能】
印刷プレビュー時に、スクリーンの解像度を出力するプリンタの解像度にスケーリングします。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**PreviewInit**(HDC pDC, HDC dDC)
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**PreviewInit**(pDC, dDC: HDC)

【引数】

名称	内容
phDC	出力するプリンタのデバイスコンテキスト
dhDC	プレビューするスクリーンのデバイスコンテキスト

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
印刷プレビュー時に、スクリーンの解像度を出力するプリンタの解像度にスケーリングします。

PrintAbortDoc（イメージキットコントロール／PrintDraw メソッド）

【機能】
現在の印刷ジョブを終了します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**PrintAbortDoc**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**PrintAbortDoc**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
現在の印刷ジョブを終了し、最後に **PrintStartDoc** メソッドを呼び出した後に描画したすべての情報を消去します。
PrintAbortDoc メソッドは、**PrintStartDoc**,**PrintStartPage**,**PrintEndPage**,**PrintEndDoc** メソッドと共に使用し、**PrintStartPage** と **PrintEndPage** メソッドの間に記述します。
ただし、**ButtonName**,**Caption**,**Message** プロパティなどを設定して印刷中止のダイアログボックスを表示する場合は、実行する必要はありません。

印刷ジョブを終了する例：
(1)C++Builder

```
bool Ret;
int i;
bool bAbort = false;

VImageKit1->PrintDraw->PrintFileName = "Default";
if (VImageKit1->PrintDraw->PrintCreateDC(vikPrintFileName) == false) return;

VImageKit1->PrintDraw->DocName = "ImageKit Print Sample";
if (VImageKit1->PrintDraw->PrintStartDoc() != false) {
   for (i = 0; i < 100; i++) {
      VImageKit1->PrintDraw->PrintStartPage();

      //イメージ・テキストなどの描画

      if (i == 50) {
         VImageKit1->PrintDraw->PrintAbortDoc();
         bAbort = true;
      }
      VImageKit1->PrintDraw->PrintEndPage();
      if (bAbort) break;
   }
   VImageKit1->PrintDraw->PrintEndDoc();
}
VImageKit1->PrintDraw->PrintDeleteDC();
```

(2)Delphi

```
Ret: Boolean;
i: Integer;
bAbort: Boolean;

VImageKit1.PrintDraw.PrintFileName := 'Default';
if VImageKit1.PrintDraw.PrintCreateDC(vikPrintFileName) = False then Exit;
```

```
VImageKit1.PrintDraw.DocName := 'ImageKit Print Sample';
if (VImageKit1.PrintDraw.PrintStartDoc <> False) then
begin
   bAbort := False;
   for i := 0 to 99 do
   begin
      VImageKit1.PrintDraw.PrintStartPage;

      //イメージ・テキストなどの描画

      if i = 50 then
      begin
         VImageKit1.PrintDraw.PrintAbortDoc;
         bAbort := True;
      end;
      VImageKit1.PrintDraw.PrintEndPage;
      if bAbort then Break;
   end;
   VImageKit1.PrintDraw.PrintEndDoc;
end;
VImageKit1.PrintDraw.PrintDeleteDC;
```

PrintCreateDC(イメージキットコントロール／PrintDraw メソッド)

【機能】
デバイスコンテキストを作成します。

【書式】
(1)C++Builder　[*bool* =]*imagekitcontrolname*->**PrintDraw**->**PrintCreateDC**(TVIkPrintMode Mode)
(2)Delphi　[*Boolean* =]*imagekitcontrolname*.**PrintDraw**.**PrintCreateDC**(Mode: TVIkPrintMode)

【引数】

名称	内容
Mode	デバイスコンテキストを作成する方法

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Modeが vikPrinterNameの場合
PrinterName プロパティに設定されたプリンタ名からデバイスコンテキストを作成します。
Modeが vikPrintFileNameの場合
PrintFileName プロパティに設定されたファイルからデバイスコンテキストを作成します。
Modeが vikDevModeの場合
DevMode プロパティに設定された DEVMODE 構造体のハンドルと **DevNames** プロパティに設定された DEVNAMES 構造体のハンドルからデバイスコンテキストを作成します。

通常使うプリンタの印刷設定をそのまま使用する場合は、Mode を vikPrintFileName にして **PrintFileName** プロパティを"Default"(アルファベットの大小文字関係なし)に設定してください。
成功すると **Handle** プロパティにデバイスコンテキストが作成されます。また、**XResolution** プロパティと **YResolution** プロパティにデバイスコンテキストの解像度が設定されます。なお、取得したデバイスコンテキストは **PrintDeleteDC** メソッドで削除します。

拡張機能として、**PrintFileName** プロパティにプリンタの設定情報を保存したファイル名にプラスして印刷部数、印刷の向き、用紙サイズを順番に設定すると、保存した項目の該当する部分を変更して印刷することができます。(後ろの項目は省略可、順番を入れ替えることは不可)
設定する場合はそれぞれの項目をセミコロン(;)で区切ります。
設定ファイルに保存された情報を有効にしたい場合は、それぞれの該当する項目に 0 を設定します。

印刷の向き:()内の説明は WindowsAPI で使用する定数と同じ意味です。
　1:縦(DMORIENT_PORTRAIT)　　2:横(DMORIENT_LANDSCAPE)
用紙サイズ:()内の説明は WindowsAPI で使用する定数と同じ意味です。
　8:A3(DMPAPER_A3)　　9:A4(DMPAPER_A4)
12:B4(DMPAPER_B4)　　13:B5(DMPAPER_B5)
上記以外の用紙サイズについては、ご利用のコンテナの WindowsAPI に関連する部分を参考に設定してください。

設定ファイルを"IkPrint.Ik"とした場合(Delphi)
1)通常の場合
```
VImageKit1.PrintDraw.PrintFileName := 'C:¥Test¥IkPrint.Ik';
```
2)拡張機能を使用した場合
A)印刷部数を 3、印刷の向きを横、用紙サイズを B5 にする
```
VImageKit1.PrintDraw.PrintFileName := 'C:¥Test¥IkPrint.Ik;3;2;13';
```
B)印刷部数と用紙サイズを設定ファイルから読み出し、印刷の向きを横にする
```
VImageKit1.PrintDraw.PrintFileName := 'C:¥Test¥IkPrint.Ik;0;2;0'; or VImageKit1.PrintDraw.PrintFileName :=
'C:¥Test¥IkPrint.Ik;0;2';
```

プリンタ設定ファイル(IkPrn.Ik)を使用して印刷するコード例:
(1)C++Builder

```
bool Ret;

VImageKit1->PrintDraw->PrintFileName = "C:¥¥Test¥¥IkPrn.Ik";
Ret = VImageKit1->PrintDraw->PrintCreateDC(vikPrintFileName);
if (Ret == false) return;

VImageKit1->PrintDraw->DocName = "ImageKit Print Sample";
if (VImageKit1->PrintDraw->PrintStartDoc() != false) {
  VImageKit1->PrintDraw->PrintStartPage();

  //イメージ・テキストなどの描画

  VImageKit1->PrintDraw->PrintEndPage();
  VImageKit1->PrintDraw->PrintEndDoc();
}
VImageKit1->PrintDraw->PrintDeleteDC();
```

(2)Delphi

```
Ret: Boolean;

VImageKit1.PrintDraw.PrintFileName := 'C:¥Test¥IkPrn.Ik';
Ret := VImageKit1.PrintDraw.PrintCreateDC(vikPrintFileName);
if Ret = False then Exit;

VImageKit1.PrintDraw.DocName := 'ImageKit Print Sample';
if (VImageKit1.PrintDraw.PrintStartDoc <> False) then
begin
  VImageKit1.PrintDraw.PrintStartPage;

  //イメージ・テキストなどの描画

  VImageKit1.PrintDraw.PrintEndPage;
  VImageKit1.PrintDraw.PrintEndDoc;
end;
VImageKit1.PrintDraw.PrintDeleteDC;
```

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikPrinterName, ikPrintFileName, ikDevMode)。

【ImageKit7/8/9 ActiveX/VCL との違い】
当メソッドが成功すると **XResolution** プロパティと **YResolution** プロパティにデバイスコンテキストの解像度が設定されます。

PrintDeleteDC（イメージキットコントロール／PrintDraw メソッド）

【機能】
デバイスコンテキストを削除します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**PrintDeleteDC**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**PrintDeleteDC**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
印刷処理が終了し、デバイスコンテキストが不要になった段階で実行します。
印刷コードについては **PrintCreateDC**,**PrintDialog** メソッドを参照してください。

PrintDialog(イメージキットコントロール／PrintDraw メソッド)

【機能】
デバイスコンテキストを作成します。

【書式】
(1)C++Builder　[*bool* =]*imagekitcontrolname*->**PrintDraw**->**PrintDialog**()
(2)Delphi　[*Boolean* =]*imagekitcontrolname*.**PrintDraw**.**PrintDialog**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
印刷ダイアログを表示し、設定した情報でデバイスコンテキストを作成します。
GetDevModeHandle メソッドを実行し、**DevMode** プロパティと **DevNames** プロパティを取得すると、印刷ダイアログを表示する際にプリンタ名や印刷条件を操作できます。通常使うプリンタの印刷設定をそのまま使用する場合は、事前に **GetDevModeHandle** メソッドを実行する必要はありません。
Collate,**Copies**,**FromPage**,**ToPage**,**MaxPage**,**MinPage**,**Options**,**PrintRange**,**PrintToFile** プロパティは印刷ダイアログを表示する際の初期値として使用され、処理終了後には設定した情報が返されます。
成功すると **Handle** プロパティにプリンタのデバイスコンテキストが設定され、**DevMode** プロパティと **DevNames** プロパティが示す情報が更新されます。また、**XResolution** プロパティと **YResolution** プロパティにデバイスコンテキストの解像度が設定されます。なお、取得したデバイスコンテキストは **PrintDeleteDC** メソッドで削除し、**DevMode** プロパティと **DevNames** プロパティは **ReleaseDevModeHandle** メソッドで解放します。
PrintDialog メソッドは以前の ImageKit で提供していたプリントコントロールの **PrintDlg** メソッドに相当します。

印刷するコード例:
(1)C++Builder

```
bool Ret;

VImageKit1->PrintDraw->ClearProperty();
VImageKit1->PrintDraw->Copies = 1;
Ret = VImageKit1->PrintDraw->PrintDialog();
if (Ret == false) return;

VImageKit1->PrintDraw->DocName = "ImageKit Print Sample";
if (VImageKit1->PrintDraw->PrintStartDoc() != false) {
  VImageKit1->PrintDraw->PrintStartPage();

  //イメージ・テキストなどの描画

  VImageKit1->PrintDraw->PrintEndPage();
  VImageKit1->PrintDraw->PrintEndDoc();
}
VImageKit1->PrintDraw->PrintDeleteDC();
VImageKit1->PrintDraw->ReleaseDevModeHandle();
```

(2)Delphi

```
Ret: Boolean;

VImageKit1.PrintDraw.ClearProperty;
VImageKit1.PrintDraw.Copies := 1;
Ret := VImageKit1.PrintDraw.PrintDialog;
```

(2)Delphi

```
if Ret = False then Exit;
```

```
VImageKit1.PrintDraw.DocName := 'ImageKit Print Sample';
if (VImageKit1.PrintDraw.PrintStartDoc <> False) then
begin
  VImageKit1.PrintDraw.PrintStartPage;

  //イメージ・テキストなどの描画

  VImageKit1.PrintDraw.PrintEndPage;
  VImageKit1.PrintDraw.PrintEndDoc;
end;
VImageKit1.PrintDraw.PrintDeleteDC;
VImageKit1.PrintDraw.ReleaseDevModeHandle;
```

【ImageKit7/8/9 ActiveX/VCL との違い】

当メソッドが成功すると **DevMode** プロパティと **DevNames** プロパティが示す情報が更新され、**XResolution** プロパティと **YResolution** プロパティにデバイスコンテキストの解像度が設定されます。

PrintEndDoc（イメージキットコントロール／PrintDraw メソッド）

【機能】
印刷を終了します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**PrintEndDoc**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**PrintEndDoc**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
PrintEndDoc メソッドは、**PrintStartDoc**,**PrintStartPage**,**PrintEndPage** メソッドと共に使用します。
印刷コードについては **PrintCreateDC**,**PrintDialog** メソッドを参照してください。

PrintEndPage(イメージキットコントロール／PrintDraw メソッド)

【機能】
ページ単位の印刷を終了して、改ページを行います。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**PrintEndPage**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**PrintEndPage**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
PrintEndPage メソッドは、**PrintStartDoc**,**PrintStartPage**,**PrintEndDoc** メソッドと共に使用します。
印刷コードについては **PrintCreateDC**,**PrintDialog** メソッドを参照してください。

PrintStartDoc（イメージキットコントロール／PrintDraw メソッド）

【機能】
印刷を開始します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**PrintStartDoc**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**PrintStartDoc**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Caption,**ButtonName**,**Message** プロパティが設定されていない場合は、印刷中止のダイアログボックスは表示されません。
PrintStartDoc メソッドは、**PrintStartPage**,**PrintEndPage**,**PrintEndDoc** メソッドと共に使用します。
印刷コードについては **PrintCreateDC**,**PrintDialog** メソッドを参照してください。

PrintStartPage(イメージキットコントロール／PrintDraw メソッド)

【機能】
ページ単位の印刷を開始します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**PrintStartPage**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**PrintStartPage**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
PrintStartPage メソッドは **PrintStartDoc**,**PrintEndPage**,**PrintEndDoc** メソッドと共に使用します。
印刷コードについては **PrintCreateDC**,**PrintDialog** メソッドを参照してください。

Rectangle(イメージキットコントロール／PrintDraw メソッド)

【機能】
描画先オブジェクトに矩形を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Rectangle**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Rectangle**(NativeUInt DeviceValue, const TRect &ARect, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Rectangle**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**Rectangle**(void * DeviceValue, const TRect &ARect, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Rectangle**(DeviceValue: THandle; ALeft, ATop, ARight, ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Rectangle**(DeviceValue: THandle; const ARect: TRect; Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Rectangle**(DeviceValue: Pointer; ALeft, ATop, ARight, ABottom: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**Rectangle**(DeviceValue: Pointer; const ARect: TRect; Integer; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル(DeviceMode による)
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
DeviceMode	描画対象(vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル)

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ALeft,ATop,ARight,ABottom もしくは ARect で指定した座標を基に矩形を描画します。アウトラインは **PenStyle** プロパティの値で描画され、内部は **BrushStyle** プロパティの値で塗りつぶされます。(対象はスクリーン、プリンタ、メモリハンドル)
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**BrushStyle**,**BrushColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。**BackColor** プロパティは **Transparent** プロパティが False でペンが実線以外、あるいはブラシがハッチパターンの場合に有効です。

ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】

・列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikPrinter, ikMemoryHandle)。

・TRect 型を渡すメソッドが実装されました。

ReleaseDevModeHandle（イメージキットコントロール／PrintDraw メソッド）

【機能】
DEVMODE 構造体のハンドルと DEVNAMES 構造体のハンドルを解放します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw->ReleaseDevModeHandle**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw.ReleaseDevModeHandle**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
GetDevModeHandle メソッドや **PrintDialog** メソッドで取得した **DevMode** プロパティと **DevNames** プロパティが占有するメモリを解放します。

【ImageKit7/8/9 ActiveX/VCL との違い】
当メソッドが成功すると **DevMode** プロパティに加え **DevNames** プロパティが占有するメモリを解放します。

RoundRect（イメージキットコントロール／PrintDraw メソッド）

【機能】
描画先オブジェクトに角が丸の矩形を描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**RoundRect**(NativeUInt DeviceValue, int ALeft, int ATop, int ARight, int ABottom, int X, int Y, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**RoundRect**(NativeUInt DeviceValue, const TRect &ARect, int X, int Y, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**RoundRect**(void * DeviceValue, int ALeft, int ATop, int ARight, int ABottom, int X, int Y, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**RoundRect**(void * DeviceValue, const TRect &ARect, int X, int Y, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**RoundRect**(DeviceValue: THandle; ALeft, ATop, ARight, ABottom, X, Y: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**RoundRect**(DeviceValue: THandle; const ARect: TRect; X, Y: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**RoundRect**(DeviceValue: Pointer; ALeft, ATop, ARight, ABottom, X, Y: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**RoundRect**(DeviceValue: Pointer; const ARect: TRect; X, Y: Integer; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
ALeft,ATop	矩形の左上の x,y 座標
ARight,ABottom	矩形の右下の x,y 座標
ARect	矩形の x,y 座標
X,Y	丸い角を描画するための楕円の幅と高さ
DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

※ALeft,ATop,ARight,ABottom もしくは ARect のどちらかを使用します。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
ALeft,ATop,ARight,ABottom もしくは ARect で指定した座標を基に矩形を描画し、その角が指定の楕円のパラメータによって丸められます。アウトラインは **PenStyle** プロパティの値で描画され、内部は **BrushStyle** プロパティの値で塗りつぶされます。（描画対象はスクリーン、プリンタ、メモリハンドル）
描画するには **PenWidth**,**PenStyle**,**PenMode**,**PenColor**,**BrushStyle**,**BrushColor**,**Transparent**,**BackColor** プロパティに値を設定する必要があります。**BackColor** プロパティは **Transparent** プロパティが False でペンが実線以外、あるいはブラシがハッチパターンの場合に有効です。

ALeft,ATop,ARight,ABottom(もしくはARect)をピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
ALeft,ATop,ARight,ABottom(もしくはARect)を 0.1mm単位として扱う場合

DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました（ActiveX は ikScreen, ikPrinter, ikMemoryHandle）。
・TRect 型を渡すメソッドが実装されました。

SaveDevModeHandle（イメージキットコントロール／PrintDraw メソッド）

【機能】
DEVMODE 構造体のハンドルと DEVNAMES 構造体のハンドルをファイルに保存します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw->SaveDevModeHandle**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw.SaveDevModeHandle**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
GetDevModeHandle メソッドで取得した **DevMode** プロパティと **DevNames** プロパティを **PrintFileName** プロパティに設定されたファイルに保存します。保存されるファイル形式は **SavePrinterInfo** メソッドで保存されるファイル形式と同じです。
そのため、保存したファイルは **GetDevModeHandle**,**PrintCreateDC**,**SavePrinterInfo** メソッドで使用可能です。
DEVMODE 構造体と DEVNAMES 構造体については WindowsAPI 関連の書籍などをご覧ください。
PrintFileName プロパティにファイル名のみを設定すると、プリンタ設定ファイルはカレントフォルダに保存されます。

【ImageKit7/8/9 ActiveX/VCL との違い】
当メソッドが成功すると **DevMode** プロパティに加え **DevNames** プロパティが示す情報をファイルに保存します。

SavePrinterInfo（イメージキットコントロール／PrintDraw メソッド）

【機能】
プリンタ設定ダイアログで設定した情報をファイルに保存します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**SavePrinterInfo**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**SavePrinterInfo**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
プリンタ設定ダイアログを表示して、設定した情報を **PrintFileName** プロパティに設定されたファイルに保存します。その際に **PrintFileName** プロパティ+″.PrnTxt″という名称のファイルに、テキストとしてプリンタ名・出力先名を保存します。内訳は 1 レコード目がプリンタ名、2 レコード目が出力先名となります。レコードのデリミタ（区切り）は{CR}{LF}です。
（プリンタ設定ダイアログを終了する際に OK を選択した場合、上記の 2 つのファイルを保存し、戻り値 0 以外を返します。）
保存した設定ファイルは、**GetDevModeHandle**,**PrintCreateDC** メソッドで使用します。（テキストとして保存したファイルは使用しません。）
PrintFileName プロパティにファイル名のみを設定すると、プリンタ設定ファイルはカレントフォルダに保存されます。

SavePrinterInfo メソッドは以前の ImageKit で提供していたプリントコントロールの **SetPrint** メソッドに相当します。

SetDefaultPrinter（イメージキットコントロール／PrintDraw メソッド）

【機能】
通常使うプリンタ（デフォルトプリンタ）を設定します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**SetDefaultPrinter**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**SetDefaultPrinter**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Windows のプリンタフォルダに登録されているプリンタを通常使うプリンタとして設定できます。
PrinerName プロパティに **EnumPrinters** メソッドで取得したリストに含まれるプリンタ名を設定し、**SetDefaultPrinter** メソッドを実行します。

SetDevModeInfo(イメージキットコントロール／PrintDraw メソッド)

【機能】
DEVMODE 構造体のハンドルが指す内容を更新します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**PrintDraw**->**SetDevModeInfo**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**SetDevModeInfo**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
SetDevModeInfo メソッドを実行するには、**GetDevModeHandle** メソッドで **DevMode** プロパティを取得する必要があります。**Collate**,**ColorMode**,**Copies**,**CustomPaperHeight**,**CustomPaperWidth**,**Duplex**,**Orientation**,**PaperBin**,**PaperSize**,**XResolution**,**YResolution**,**Zoom** プロパティの情報を **DevMode** プロパティに反映させます。各プロパティに 0 を設定した場合は、何の変更も行いません。(**Collate** は除く)
ただし、反映させた結果についてはプリンタドライバに依存するため、その後で印刷を行っても効果がまったくなかったり、異なる設定を行っても印刷結果が同じになることがあります。詳しくはプリンタドライバのマニュアルなどを参照してください。

コード例:
(1)C++Builder

```
bool Ret;

VImageKit1->PrintDraw->PrinterName = "EPSON LP-8200C";
VImageKit1->PrintDraw->PrintFileName = "";
Ret = VImageKit1->PrintDraw->GetDevModeHandle();
if (Ret == false) return;

//GetDevModeHandle 内部で実行されるためコメントに
//Ret = VImageKit1->PrintDraw->GetDevModeInfo();
if (VImageKit1->PrintDraw->Copies != 3)
  VImageKit1->PrintDraw->Copies = 3; //印刷部数を 3 に
Ret = VImageKit1->PrintDraw->SetDevModeInfo();

Ret = VImageKit1->PrintDraw->PrintCreateDC(vikDevMode);
if (Ret != false)
{
  //印刷処理・・・
  Ret = VImageKit1->PrintDraw->PrintDeleteDC();
}
Ret = VImageKit1->PrintDraw->ReleaseDevModeHandle();
```

(2)Delphi

```
Ret: Boolean;

VImageKit1.PrintDraw.PrinterName := 'EPSON LP-8200C';
VImageKit1.PrintDraw.PrintFileName := '';
Ret := VImageKit1.PrintDraw.GetDevModeHandle;
if Ret = False then Exit;

//GetDevModeHandle 内部で実行されるためコメントに
//Ret := VImageKit1.PrintDraw.GetDevModeInfo;
if VImageKit1.PrintDraw.Copies <> 3 then
```

```
   VImageKit1.PrintDraw.Copies := 3; //印刷部数を 3 に
Ret := VImageKit1.PrintDraw.SetDevModeInfo;

Ret := VImageKit1.PrintDraw.PrintCreateDC(vikDevMode);
if Ret <> False then
begin
   //印刷処理・・・
   Ret := VImageKit1.PrintDraw.PrintDeleteDC;
end;
Ret := VImageKit1.PrintDraw.ReleaseDevModeHandle;
```

SetPixel(イメージキットコントロール／PrintDraw メソッド)

【機能】
描画先オブジェクトの指定したピクセルに指定したカラーを設定します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**SetPixel**(NativeUInt DeviceValue, int X, int Y, TColor AColor, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**SetPixel**(void * DeviceValue, int X, int Y, TColor AColor, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**SetPixel**(DeviceValue: THandle; X, Y: Integer; AColor: TColor; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**SetPixel**(DeviceValue: Pointer; X, Y: Integer; AColor: TColor; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル(DeviceMode による)
X,Y	ピクセルの座標
AColor	ピクセルの新しいカラー
DeviceMode	描画対象(vikScreen:スクリーン、vikMemoryHandle:メモリハンドル)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
X,Y で指定したピクセルに AColor で指定したカラーを設定します。
指定のカラーをデバイスが正確に再現できない場合には、近似のカラーが用いられます。
(対象はスクリーン、メモリハンドル)
DeviceMode が vikScreen もしくは vikMemoryHandle の場合に有効になります。X,Y の値はピクセル単位で設定してください。
AColor には色定数(clRed など)や **RGB**(Red,Green,Blue)として求めた値などを設定してください。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikScreen, ikMemoryHandle)。

TextOut（イメージキットコントロール／PrintDraw メソッド）

【機能】
描画先オブジェクトにテキストを描画します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**TextOut**(NativeUInt DeviceValue, int X, int Y, TVIkOutPutDeviceMode DeviceMode)
[*bool =*]*imagekitcontrolname*->**PrintDraw**->**TextOut**(void * DeviceValue, int X, int Y, TVIkOutPutDeviceMode DeviceMode)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**TextOut**(DeviceValue: THandle; X, Y: Integer; DeviceMode: TVIkOutPutDeviceMode)
[*Boolean =*]*imagekitcontrolname*.**PrintDraw**.**TextOut**(DeviceValue: Pointer; X, Y: Integer; DeviceMode: TVIkOutPutDeviceMode)

【TVIkOutPutDeviceMode 型】
ユニット
IkInit
type
TVIkOutPutDeviceMode = (vikScreen, vikPrinter, vikMemoryHandle);

【引数】

名称	内容
DeviceValue	デバイスコンテキストもしくはラスタイメージのメモリハンドル（DeviceMode による）
X,Y	描画開始位置の座標
DeviceMode	描画対象（vikScreen:スクリーン、vikPrinter:プリンタ、vikMemoryHandle:メモリハンドル）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
X,Y で指定した座標からテキストを描画します。（対象はスクリーン、プリンタ、メモリハンドル）
描画するには **TextColor1**,**TextColor2**,**FontName**,**FontSize**,**Transparent**,**Direction**,**CharAngle**,**CharExtra**,**Text**,**RotateString** プロパティに値を設定する必要があります。
CharExtra プロパティに 1 以上の値が設定されていた場合は文字の間隔を考慮します。
RotateString プロパティが False の場合、**CharAngle** プロパティは 0,90,180,270 以外は無効です。文字列の方向に関わらず、X,Y から常に右方向に描画されます。
RotateString プロパティが True の場合、**Direction** プロパティは無効です。X,Y を始点として文字列を回転させた方向に描画されます。
Text プロパティに改行コードを含む複数行の文字列を設定しても当メソッドでは正しく出力されないため、単一行に切り出して複数回出力を行うか、**DrawText** メソッドをご使用ください。

X,Y,**CharExtra**プロパティをピクセル単位として扱う場合
DeviceMode が vikScreen もしくは vikMemoryHandle
X,Y,**CharExtra**プロパティを 0.1mm単位として扱う場合
DeviceMode が vikPrinter

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました（ActiveX は ikScreen, ikPrinter, ikMemoryHandle）。

Scan（イメージキットコントロール／カスタム階層プロパティ）

【機能】
TWAIN 対応のスキャンデバイスからイメージを取り込む機能を提供します。

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
AdjustGamma	開発者作成 UI で取り込む時のガンマ補正係数
AutoBright	開発者作成 UI で取り込む時の明るさを自動で処理するかどうかを設定
BitDepth	開発者作成 UI で取り込む時の画素ビット数
BitDepthReduction	開発者作成 UI で取り込む時の色深度の減少方法
Border	開発者作成 UI で取り込む時の境界補正
BorderColor	開発者作成 UI で取り込む時の境界補正色
BorderDetection	開発者作成 UI で取り込む時の自動領域切り出し
Brightness	開発者作成 UI で取り込む時の明るさを設定
Cancel	処理の中止の設定
ColorBWRatio	開発者作成 UI で取り込む時の白黒原稿の中に含まれるカラー画素の割合
Compression	ファイルおよびメモリ転送時の圧縮方法
Contrast	開発者作成 UI で取り込む時のコントラスト
Deskew	開発者作成 UI で取り込む時のデスキュー（傾き補正）
DropoutColor	開発者作成 UI で取り込む時のドロップアウトカラー
DataSourceIndex	List メソッドで取得されるデータソースのカレントを示すインデックス
DataSourceName	取り込みを行うデータソース名
DataSourceNameList	List メソッドで取得されるデータソースのリスト
DynamicThreshold	開発者作成 UI で取り込む時のダイナミックスレッシュホールド
ExtUiMode	拡張ユーザインタフェース（メーカ提供 UI）モードの設定
FileFormat	ファイル転送時の保存するファイル形式
FileName	ファイル転送時およびメモリ転送の圧縮モード時に保存するファイル名
FocusPosition	開発者作成 UI で取り込む時の焦点位置
FpuException	FPU 例外をすべて無効にするかどうかを設定
Gamma	開発者作成 UI で取り込む時のガンマ
Halftone	開発者作成 UI で取り込む時のハーフトーン名
Highlight	開発者作成 UI で取り込む時のハイライト
IgnoreBackColor	開発者作成 UI で取り込む時、原稿の背景色を無視するかどうかを設定
ImageFilter	開発者作成 UI で取り込む時のイメージフィルタ
ImageMerge	開発者作成 UI で取り込む時の両面合成（ADF 両面時）
Indicator	開発者作成 UI で取り込む時のインジケータの有無
InformationFileName	メーカ提供 UI の設定情報を保存するファイル名
JpegQuality	ファイルおよびメモリ転送時における圧縮方法が JPEG の場合の品質係数
Manufacturer	Select メソッドや GetDataSourceInfo メソッドで取得されるデータソースの製造者名
MoireFilter	開発者作成 UI で取り込む時のモアレ除去フィルタ
NoiseFilter	開発者作成 UI で取り込む時のノイズ除去フィルタ
Orientation	開発者作成 UI で取り込む時の用紙の向き、または画像の回転角度
OverScan	開発者作成 UI で取り込む時のオーバースキャンの取り扱い
PageCount	開発者作成 UI で取り込む時のページ数
PaperSize	開発者作成 UI で取り込む時の用紙サイズ
PixelType	開発者作成 UI で取り込む時のイメージの種類
ProductFamily	Select メソッドや GetDataSourceInfo メソッドで取得されるデータソースの製品ファミリー名
ProductName	Select メソッドや GetDataSourceInfo メソッドで取得されるデータソースの製品名
ProtocolMajor	Select メソッドや GetDataSourceInfo メソッドで取得される TWAIN のメジャーバージョン
ProtocolMinor	Select メソッドや GetDataSourceInfo メソッドで取得される TWAIN のマイナーバージョン
RangeCurrent	GetCapRange メソッドで取得される現在値
RangeDefault	GetCapRange メソッドで取得されるデフォルト値
RangeMax	GetCapRange メソッドで取得される最大値
RangeMin	GetCapRange メソッドで取得される最小値
RangeStep	GetCapRange メソッドで取得されるステップ値

RectBottom	開発者作成 UI で取り込む時の取込下位置
RectLeft	開発者作成 UI で取り込む時の取込左位置
RectRight	開発者作成 UI で取り込む時の取込右位置
RectTop	開発者作成 UI で取り込む時の取込上位置
RemoveHole	開発者作成 UI で取り込む時のパンチ穴の取り扱い
RotateBack	開発者作成 UI で取り込む時の両面スキャン時の裏面回転
Rotation	開発者作成 UI で取り込む時の画像の回転角度
ScanCount	取り込んだイメージの数
ScanMode	開発者作成 UI で取り込む時の取り込みモード
ScanningSpeed	開発者作成 UI で取り込む時の読み取り速度
Shadow	開発者作成 UI で取り込む時のシャドウ
Sharpness	開発者作成 UI で取り込む時のシャープネス
SkipBlankPage	開発者作成 UI で取り込む時の白紙ページの取り扱い
SkipBlankThreshold	開発者作成 UI で取り込む時の白紙ページに含まれるしきい値の割合や白紙スキップのしやすさ
SourceMajor	Select メソッドや GetDataSourceInfo メソッドで取得されるデータソースのメジャーバージョン
SourceMinor	Select メソッドや GetDataSourceInfo メソッドで取得されるデータソースのマイナーバージョン
SourceVersionInfo	Select メソッドや GetDataSourceInfo メソッドで取得されるデータソースのバージョン情報
TextEnhancement	開発者作成 UI で取り込む時のテキストエンハンスメント
Threshold	開発者作成 UI で取り込む時のスレッシュホールド(しきい値)
TransferMode	イメージの転送方法
UiMode	取り込む時のユーザインタフェース(メーカ提供 UI or 開発者作成 UI)の設定
UnitFlag	開発者作成 UI で取り込む時の取り込み単位を DS の機能に任せるか IK10 で考慮するか
UnitMode	開発者作成 UI で取り込む時の取り込み単位、および実際に取り込んだ単位
XResolution	X 方向の解像度
XScaling	開発者作成 UI で取り込む時の X 方向のスケーリングの比率
YResolution	Y 方向の解像度
YScaling	開発者作成 UI で取り込む時の Y 方向のスケーリングの比率

【ImageKit7/8/9/10 ActiveX との違い】

削除されたプロパティ: DsNameCount, HalfToneList

変更されたプロパティ:

DsName --> DataSourceName
DsNameList --> DataSourceNameList
HalfTone --> Halftone

●メソッド一覧(アルファベット順)

カスタムメソッド	内容
ClearProperty	Scan プロパティの初期化
CloseDataSource	データソースをクローズ
Execute	データソースからの取り込み
FreeTwain	TWAIN DLL の解放
GetBitDepth	サポートしている画素ビット数の取得
GetCapEnumToFloat	指定した項目の設定可能値の取得(整数および実数型)
GetCapEnumToString	指定した項目の設定可能値の取得(文字列型)
GetCapRange	指定した項目の範囲の取得
GetDataSourceInfo	指定したデータソースの情報を取得
GetMinimumSize	データソースから取り込める最小サイズの取得
GetPhysicalSize	データソースから取り込める最大物理サイズの取得
Initialize	TWAIN の初期化
IsCapSupported	指定した項目がサポートされているかどうかを確認
IsOpenDataSource	データソースがオープンされているかどうかを確認
List	データソースの列挙
LoadTwain	TWAIN DLL のロード
OpenDataSource	データソースをオープン
Select	データソース(ドライバ)の選択
Terminate	TWAIN の終了処理

【ImageKit7/8/9/10 ActiveX との違い】
変更されたメソッド:
CloseDS --> CloseDataSource
Exec --> Execute
GetCapEnum --> GetCapEnumToFloat, GetCapEnumToString
GetDSInfo --> GetDataSourceInfo
IsOpenDS --> IsOpenDataSource
OpenDS --> OpenDataSource

●ImageKit10 で扱うイメージ取込の分類表

取込装置	取込イメージ	取込方法
イメージスキャナ(原稿台)	シングルイメージ	開発者作成 UI/メーカ提供 UI
イメージスキャナ(ADF)	マルチイメージ	開発者作成 UI/メーカ提供 UI
デジタルカメラ	マルチイメージ	開発者作成 UI/メーカ提供 UI
フィルムスキャナ	マルチイメージ	メーカ提供 UI

●取り込みの簡単な流れ
[LoadTwain]
|
Initialize(必須)
|
GetDataSourceInfo, List, Select
|
[OpenDataSource]
|
GetBitDepth, GetCapEnumToFloat, GetCapEnumToString, GetCapRange, GetPhysicalSize, GetMinimumSize, IsCapSupported
|
Execute
|
[CloseDataSource]
|
Terminate(必須)
|
[FreeTwain]

注意:
[ ]の LoadTwain/FreeTwain と OpenDataSource/CloseDataSource は実行しなくても構いませんが、実行する時は LoadTwain/FreeTwain と OpenDataSource/CloseDataSource を対で使用してください。LoadTwain/FreeTwain と Initialize/Terminate を除く全てのメソッドは、その間であればどのタイミングでも使用可能です。
ただし、OpenDataSource/CloseDataSource を実行する場合、OpenDataSource/CloseDataSource の間に記載されているメソッドは OpenDataSource を実行した後に使用しなければなりません。

AdjustGamma(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のガンマ補正係数を設定します

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**AdjustGamma** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.AdjustGamma** [= *Smallint*]

【設定値】

値	説明
0	係数 1.0
1	係数 1.8

デフォルトは 0 です。

【解説】
エプソン製スキャナ用ドライバご利用時に有効です。0 よりも 1 の方が画像が明るくなります。
UiMode プロパティが vikScanNONUI の場合に有効です。
データソースが機能をサポートしているかどうかについては、**IsCapSupported** メソッドで判定できます。

【値の設定】 実行時

【値の参照】 不可

AutoBright(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の自動領域切り出しの有無を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**AutoBright** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Scan.AutoBright** [= *Boolean*]

【設定値】

値	説明
True	明るさを手動で処理する
False	明るさを自動で処理する

【解説】
デフォルトは False です。**UiMode** プロパティが vikScanNONUI の場合に有効です。
データソースが機能をサポートしているかどうかについては、**IsCapSupported** メソッドや **GetCapEnumToFloat** メソッドで判定できます。データソースが TWAIN2.0 以降に対応している場合は、**GetCapEnumToFloat** メソッドサポートする値を取得できます。当プロパティの値が False またはデータソースが機能をサポートしてない場合は、**Brightness** プロパティの値が有効になります。

【値の設定】 実行時

【値の参照】 不可

BitDepth(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の画素ビット数を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**BitDepth** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**BitDepth** [= *Smallint*]

【設定値】
PixelType プロパティが vikScanPixelBW の場合は 1。
PixelType プロパティが vikScanPixelGray の場合は 4,8,12,14,16 のいずれか。
PixelType プロパティが vikScanPixelRGB の場合は 24,36,42,48 のいずれか。
PixelType プロパティが vikScanPixelPalette の場合は 4,8 のいずれか。
PixelType プロパティが vikScanPixelAutomation の場合は 24。

【解説】
UiMode プロパティが vikScanNONUI の場合に有効です。
ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
設定可能な値については、**GetBitDepth** メソッドで取得できます。
12,14,16 もしくは 36,42,48 を設定して取り込みを行う場合は、**TranferMode** プロパティを vikScanMemory に設定してください。データソースでサポートしている値にも関わらずエラーとなる場合は、0 を設定してください。0 の場合は **PixelType** プロパティに応じたデフォルト値を使用します。

【値の設定】 実行時

【値の参照】 実行時

BitDepthReduction(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のイメージの色深度の減少方法を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**BitDepthReduction** [= *TVIkBitDepthReduction*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**BitDepthReduction** [= *TVIkBitDepthReduction*]

【TVIkBitDepthReduction 型】
ユニット
ImageKit
type
TVIkBitDepthReduction = (vikBitDepthReductionThreshold, vikBitDepthReductionHalftone, vikBitDepthReductionCustomHalftone, vikBitDepthReductionDiffusion, vikBitDepthReductionDynamicThreshold);

【設定値】

値	説明
vikBitDepthReductionThreshold	しきい値
vikBitDepthReductionHalftone	ハーフトーン
vikBitDepthReductionCustomHalftone	カスタムハーフトーン *1
vikBitDepthReductionDiffusion	拡散
vikBitDepthReductionDynamicThreshold	動的しきい値

*1 ImageKit10 ではサポートしておりません。

【解説】
UiMode プロパティが vikScanNONUI、**PixelType** プロパティが vikScanPixelBW の場合に有効です。
ただし、デジタルカメラからの取り込みの場合は無効です。
BitDepthReduction プロパティが vikBitDepthReductionThreshold の場合は **Threshold** プロパティ(データソースによっては **Brightness** プロパティ)が有効となり、vikBitDepthReductionHalftone の場合は **Halftone**,**Brightness**,**Contrast** プロパティが有効です。
データソースがサポートしている値は **GetCapEnumToFloat** メソッドで取得できます。仮にデータソースが **BitDepthReduction** の機能をサポートしていなくても、**Threshold** もしくは **Halftone** のどちらかをサポートしている場合は、このプロパティに vikBitDepthReductionThreshold もしくは vikBitDepthReductionHalftone を設定してください。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikBitDepthReductionThreshold, ikBitDepthReductionHalfTone, ikBitDepthReductionCustomHalfTone, ikBitDepthReductionDiffusion, ikBitDepthReductionDynamicThreshold)。
ikBitDepthReductionDynamicThreshold は ImageKit9/10 のみ。
・識別子の中に含まれる HalfTone が Halftone に変更されました。

Border(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の境界補正を行うかどうかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Border** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.Border** [= *Smallint*]

【設定値】

値	説明
エプソン製スキャナ用ドライバ(境界補正)	
0	境界補正しない
1	境界補正する
パナソニック製スキャナ用ドライバ(境界削除)	
0	境界削除しない
1	境界削除する – 自動モード(指定された幅まで原稿領域の縁を検出して補正)
2	境界削除する – 固定幅モード(指定された幅で画像の縁から一律の領域を補正)

【解説】
エプソン製スキャナ用ドライバ、パナソニック製スキャナ用ドライバご利用時に有効です。
エプソン製スキャナ用ドライバでは「境界補正」(原稿端付近に発生する背景色の写りこみや影の補正、または原稿端への枠の付加)、パナソニック製スキャナ用ドライバでは「境界削除」(読み取りやコピーにおいて発生する画像周辺の黒い縁を取り除く)という表現を用いています。
デフォルトは 0 です。**UiMode** プロパティが vkScanNONUI の場合に有効です。
1 以上の場合、**BorderColor** プロパティに設定した色で塗りつぶしが行われます。
データソースが機能をサポートしているかどうかについては、**IsCapSupported** メソッドで判定できます。

【値の設定】 実行時

【値の参照】 不可

BorderColor(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の境界補正色を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**BorderColor** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.BorderColor** [= *Smallint*]

【設定値】

値	説明
エプソン製スキャナ用ドライバ	
0	白色
1	黒色
パナソニック製スキャナ用ドライバ	
0	周辺色(原稿端の色)
1	原稿背景色
2	白色

【解説】
エプソン製スキャナ用ドライバ、パナソニック製スキャナ用ドライバご利用時に有効です。
デフォルトは 0 です。**UiMode** プロパティが vikScanNONUI、**Border** プロパティが 1 以上の場合に有効です。
補正の適用範囲はエプソン製スキャナ用ドライバ、パナソニック製スキャナ用ドライバともユーザインタフェース(メーカ提供 UI)で設定された値が使用されます。

【値の設定】 実行時

【値の参照】 不可

BorderDetection(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の自動領域切り出しの有無を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**BorderDetection** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Scan.BorderDetection** [= *Boolean*]

【設定値】

値	説明
True	自動領域切り出しを有効
False	自動領域切り出しを無効

【解説】
用紙サイズに応じて原稿領域を自動で切り出します。
UiMode プロパティが vikScanNONUI、**PaperSize** プロパティが 1000(vikScanUndefinedSize)以上の場合に有効です。
ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
データソースが機能をサポートしているかどうかについては、**IsCapSupported** メソッドで判定できます。

【値の設定】 実行時

【値の参照】 不可

Brightness,Contrast(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の明るさ・コントラストを設定します。

【書式】
※**Brightness** にて説明(**Contrast** も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Brightness** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**Brightness** [= *Smallint*]

【設定値】
−1000～1000。

Brightness は明るさ。(デフォルトは 0)
Contrast はコントラスト。(デフォルトは 0)

キヤノン製 DR スキャナ用ドライバご利用時で **PixelType** プロパティが vikScanPixelBW で **BitDepthReduction** プロパティが vikBitDepthReductionThreshold、**TextEnhancement** プロパティが 1 以上の場合は、1～255(デフォルトは 128)の範囲が有効です。

PFU 製スキャナ用ドライバご利用時で **PixelType** プロパティが vikScanPixelAutomation,vikScanPixelGrayAndBW,vikScanPixelRGBAndBW)の場合は、−127～127(デフォルトは 0)の範囲が有効です。

【解説】
UiMode プロパティが vikScanNONUI の場合に有効です。
Brightness,**Contrast** プロパティは **PixelType** プロパティが vikScanPixelBW の場合、**BitDepthReduction** プロパティの値により効果がない場合があります。(下記は効果がある場合)

PixelType	vikScanPixelBW	vikScanPixelGray,vikScanPixelRGB,vikScanPixelPalette
Brightness	**BitDepthReduction** = 0(*),1	○
Contrast	**BitDepthReduction** = 1	○

(*)しきい値の機能を明るさで表すデータソースで使用(**Threshold** をサポートしていないデータソースなど)

ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
設定可能な値については、**GetCapRange** メソッドもしくは **GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

Cancel(イメージキットコントロール／Scan プロパティ)

【機能】
イメージキットコントロールのイメージの取り込み処理を中止するかどうかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Cancel** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Scan.Cancel** [= *Boolean*]

【設定値】

値	説明
True	取り込み処理を中止する
False	取り込み処理を中止しない

【解説】
イベント(**BeforeScan**,**Scanning**,**AfterScan**)内で **Cancel** プロパティを True に設定すると、取り込み処理を中止できます。

【値の設定】 実行時

【値の参照】 不可

ColorBWRatio(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の白黒原稿の中に含まれるカラー画素の割合を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**ColorBWRatio** [= *float*]
(2)Delphi　*imagekitcontrolname*.**Scan.ColorBWRatio** [= *Single*]

【設定値】
0.01 から 50.00 までの値を%単位で設定します。
デフォルトは 1 です。

【解説】
パナソニック製スキャナ用ドライバご利用時に有効です。
値が小ければ白黒原稿はカラーとして認識される確率が高くなり、大きければ白黒として認識される確率が高くなります。
UiMode プロパティが vikScanNONUI、**PixelType** プロパティが vikScanPixelAutomation の場合に有効です。

【値の設定】 実行時

【値の参照】 不可

Compression(イメージキットコントロール／Scan プロパティ)

【機能】
ファイルおよびメモリ転送時の圧縮方法を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Compression** [= *TVIkScanCompression*]
(2)Delphi　*imagekitcontrolname*.**Scan.Compression** [= *TVIkScanCompression*]

【TVIkScanCompression 型】
ユニット
ImageKit
type
TVIkScanCompression = (vikScanNoCompress, vikScanPackbits, vikScanGroup3_1D, vikScanGroup3_1D_EOL, vikScanGroup3_2D, vikScanGroup4, vikScanJPEG, vikScanLZW, vikScanJBIG, vikScanPNG, vikScanRLE4, vikScanRLE8, vikScanBITFIELDS);

【設定値】

値	説明
vikScanNoCompress	非圧縮(デフォルト)
vikScanPackbits	PACKBITS
vikScanGroup3_1D	GROUP3-1D
vikScanGroup3_1D_EOL	GROUP3-1DEOL
vikScanGroup3_2D	GROUP3-2D
vikScanGroup4	GROUP4
vikScanJPEG	JPEG
vikScanLZW	LZW
vikScanJBIG	JBIG
vikScanPNG	PNG
vikScanRLE4	RLE4
vikScanRLE8	RLE8
vikScanBITFIELDS	BITFIELDS

vikScanGroup3_1D,vikScanGroup3_1D_EOL,vikScanGroup3_2D, vikScanGroup4 は FAX[CCITT]形式
vikScanRLE4,vikScanRLE8,vikScanBITFIELDS は BMP 形式

【解説】
開発者作成 UI で取り込む場合は、**PixelType** プロパティと **BitDepth** プロパティに適切な値を設定してください。たとえば、GROUP3/4 圧縮の場合は白黒 2 値、JPEG 圧縮の場合は 8 ビットグレースケールと 24 ビットカラーでなければなりません。
イメージの転送方法については、**TransferMode** プロパティをご覧ください。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。
データソースによってはファイル転送時の圧縮方法は非圧縮のみ有効となる場合があります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikScanNoCompress, ikScanPackbits, ikScanGroup3_1D, ikScanGroup3_1D_EOL, ikScanGroup3_2D, ikScanGroup4, ikScanJPEG, ikScanLZW, ikScanJBIG, ikScanPNG, ikScanRLE4, ikScanRLE8, ikScanBITFIELDS)。

Deskew(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のデスキュー(傾き補正)の有無を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Deskew** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.Deskew** [= *Smallint*]

【設定値】

値	説明
0	無効
1	有効
2	有効(滑らか)

【解説】
原稿の傾きを補正します。
UiMode プロパティが vikScanNONUI の場合に有効です。ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
基本的に**PaperSize**プロパティが 1000(vikScanUndefinedSize)以上の場合に有効な機能ですが、データソースによっては定型サイズでも有効な場合があります。滑らか設定はパナソニック製スキャナ用ドライバご利用時のみ有効です(メモリ転送時は非圧縮形式のみ)。
データソースが機能をサポートしているかどうかについては、**IsCapSupported** メソッドで判定できます。

【値の設定】 実行時

【値の参照】 不可

DropoutColor(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のドロップアウトカラーを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**DropoutColor** [= *TVIkDropoutColor*]
(2)Delphi　*imagekitcontrolname*.**Scan.DropoutColor** [= *TVIkDropoutColor*]

【TVIkDropoutColor 型】
ユニット
IkInit
type
TVIkDropoutColor = (vikDropoutColorRed, vikDropoutColorGreen, vikDropoutColorBlue, vikDropoutColorNone, vikDropoutColorWhite);

【設定値】

値	説明
vikDropoutColorRed	赤
vikDropoutColorGreen	緑
vikDropoutColorBlue	青
vikDropoutColorNone	なし(デフォルト)
vikDropoutColorWhite	白

【解説】
UiMode プロパティが vikScanNONUI、**PixelType** プロパティが vikScanPixelBW もしくは vikScanPixelGray の場合に有効です。だたし、デジタルカメラからの取り込みの場合は無効です。
ドロップアウトカラーを無効にする場合は、vikDropoutColorNone もしくは vikDropoutColorWhite を設定します。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭にｖが付加されました(ActiveX はikDropoutColorRed, ikDropoutColorGreen, ikDropoutColorBlue, ikDropoutColorNone, ikDropoutColorWhite)。

DataSourceIndex,DataSourceNameList(イメージキットコントロール／Scan プロパティ)

【機能】
List メソッドで取得されるデータソースのリストおよびカレントデータソースのインデックスを示します。

【書式】
※DataSourceIndex
(1)C++Builder　*imagekitcontrolname*->Scan->DataSourceIndex [= *int*]
(2)Delphi　*imagekitcontrolname*.Scan.DataSourceIndex [= *Integer*]

※DataSourceNameList
(1)C++Builder　*imagekitcontrolname*->Scan->DataSourceNameList [= *TStrings**]
(2)Delphi　*imagekitcontrolname*.Scan.DataSourceNameList [= *TStrings*]

【参照値】
DataSourceIndex はカレントデータソースのインデックス。
DataSourceNameList はデータソースのリスト。

【解説】
List メソッドが成功した場合は、リスト文字列が DataSourceNameList プロパティに、カレントデータソースのインデックスが DataSourceIndex プロパティに設定されます。
インストールされているデータソースの数は DataSourceNameList.Count で取得できます。
DataSourceIndex プロパティが-1 の場合は List メソッドが失敗もしくはメソッドを実行していない状態を示し、0 以上の場合は DataSourceNameList.Strins プロパティにアクセスする配列の添字として使用可能です。
TStrings については Delphi や C++Builder のヘルプを参照してください。

DataSourceIndex,DataSourceNameList プロパティは以前の ImageKit で提供していたスキャンコントロールの ScanDsNameCount,ScanDsNameList プロパティに相当します。

カレントデータソースに位置付ける例:
(1)C++Builder
```
bool Ret;
Ret = VImageKit1->Scan->List();
if (Ret == false || VImageKit1->Scan->DataSourceNameList->Count == 0) return;

ComboBox1->Items = VImageKit1->Scan->DataSourceNameList;
ComboBox1->ItemIndex = VImageKit1->Scan->DataSourceIndex;
```
(2)Delphi
```
Ret: Boolean;
Ret := VImageKit1.Scan.List;
if (Ret = False) or (VImageKit1.Scan.DataSourceNameList.Count = 0) then Exit;

ComboBox1.Items := VImageKit1.Scan.DataSourceNameList;
ComboBox1.ItemIndex := VImageKit1.Scan.DataSourceIndex;
```

【値の設定】 不可

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
・DataSourceNameList プロパティの名称が DsNameList から変更されました。
・DataSourceNameList プロパティの先頭にカレントデータソースは付加されないため、DataSourceIndex プロパティを使用してカレントデータソースの名称を取得します。

DataSourceName(イメージキットコントロール／Scan プロパティ)

【機能】
取り込みを行うデータソース名を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**DataSourceName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**DataSourceName** [= *string*]

【設定値】
データソース名。

【解説】
OpenDataSource,**Execute** メソッドを実行する前に設定する必要があります。
ただし、**OpenDataSource** メソッドを実行した後に **Execute** メソッドを実行する場合は設定値は無視されます。
DataSourceName プロパティに何も設定しない場合は、**Select** メソッドで選択されたデータソースが有効となります。

DataSourceName プロパティは以前の ImageKit で提供していたスキャンコントロールの **ScanDsName** プロパティに相当します。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
プロパティの名称が **DsName** から変更されました。

DynamicThreshold(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のダイナミックスレッシュホールドのレベルを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**DynamicThreshold** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.DynamicThreshold** [= *Smallint*]

【設定値】

値	説明
0	なし
1	明るい
2	少し明るい
3	標準
4	少し暗い
5	暗い

【解説】
パナソニック製スキャナ用ドライバご利用時に有効です。
ダイナミックスレッシュホールド機能を有効にすると、文字や絵がつぶれないよう、かすれないよう、最適な読み取りを行います。**UiMode** プロパティが vikScanNONUI、**PixelType** プロパティが vikScanPixelBW の場合に有効です。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。
DynamicThreshold プロパティに 1 以上を設定すると、**BitDepthReduction**,**Halftone**,**Sharpness**,**Threshold** プロパティの値は無効となります。

【値の設定】 実行時

【値の参照】 不可

ExtUiMode(イメージキットコントロール／Scan プロパティ)

【機能】
拡張ユーザインタフェースモードを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**ExtUiMode** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.ExtUiMode** [= *Smallint*]

【設定値】

値	説明
0	前回設定の UI モードで起動
1	プロフェッショナルモードで起動
2	全自動モードで起動
3	ホームモードで起動
4	全自動モードで起動(エラーメッセージを除き、UI 非表示)
5	オフィスモードで起動
7	シートフィード専用モードで起動

【解説】
エプソン製スキャナ用ドライバご利用時に有効です。
各設定値は『EPSON Scan』の UI で標準で表示されるもののみ設定可能です。
(例:ES シリーズで 2,4、GT シリーズで ADF 非接続時 5／ADF 接続時 2,4 は設定不可)
UiMode プロパティが vikScanUI,vikScanUICLOSE の場合に有効です。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

FileFormat(イメージキットコントロール／Scan プロパティ)

【機能】
ファイル転送時の保存するファイル形式を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**FileFormat** [= *TVIkScanFileFormat*]
(2)Delphi　*imagekitcontrolname*.**Scan.FileFormat** [= *TVIkScanFileFormat*]

【TVIkScanFileFormat 型】
ユニット
ImageKit
type
TVIkScanFileFormat = (vikScanFileTIFF = 0, vikScanFilePICT = 1, vikScanFileBMP = 2, vikScanFileXBM = 3, vikScanFileJFIF = 4, vikScanFileFPX = 5, vikScanFileTIFFMULTI = 6, vikScanFilePNG = 7, vikScanFileSPIFF = 8, vikScanFileEXIF = 9, vikScanFilePdf = 10, vikScanFileJp2 = 11, vikScanFileJpx = 13, vikScanFileDejavu = 14, vikScanFilePdfA = 15, vikScanFilePdfA2 = 16);

【設定値】

値	説明
vikScanFileTIFF	TIFF
vikScanFilePICT	PICT
vikScanFileBMP	BMP(デフォルト)
vikScanFileXBM	XBM
vikScanFileJFIF	JFIF(JPEG)
vikScanFileFPX	FPX
vikScanFileTIFFMULTI	TIFF MULTI
vikScanFilePNG	PNG
vikScanFileSPIFF	SPIFF
vikScanFileEXIF	EXIF
vikScanFilePdf	PDF
vikScanFileJp2	JP2(JPEG2000 Part1)
vikScanFileJpx	JPX(JPEG2000 Part2)
vikScanFileDejavu	DEJAVU
vikScanFilePdfA	PDF/A(Version 1)
vikScanFilePdfA2	PDF/A(Version 2)

【解説】
フォーマットに応じて **Compression** プロパティにも適切な値を設定します。なお、データソースによってはファイル転送時の圧縮方法は非圧縮のみ有効となる場合があります。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました。
ImageKit7 ActiveX は ikScanFileTIFF, ikScanFilePICT, ikScanFileBMP, ikScanFileXBM, ikScanFileJFIF, ikScanFileFPX, ikScanFileTIFFMULTI, ikScanFilePNG, ikScanFileSPIFF, ikScanFileEXIF で、ImageKit8 ActiveX は ikScanFilePdf , ikScanFileJp2, ikScanFileJpx, ikScanFileDejavu, ikScanFilePdfA, ikScanFilePdfA2 が追加されました。

FileName(イメージキットコントロール／Scan プロパティ)

【機能】
ファイル転送時およびメモリ転送の圧縮モード時に保存するファイル名を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**FileName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**FileName** [= *string*]

【設定値】
保存するファイル名。

【解説】
ファイル転送時(**TransferMode**プロパティがvikScanFile)およびメモリ転送の圧縮モード時(**TransferMode**プロパティがvikScanMemory,**Compression**プロパティがvikScanNoCompress以外)に読み込んだ画像を保存するファイル名を設定します(255 バイト以下)。空文字列を設定するとファイルは作成されません。ファイル転送時にはファイル名は必須です。
UiMode が vikScanNONUI で **ScanMode** が vikScanDOC の場合は設定されたファイル名で保存しますが、それ以外の場合は拡張子の前に 4 桁の連番を挿入します。(例: Image.bmp → Image0001.bmp)

【値の設定】 実行時

【値の参照】 不可

FocusPosition(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の焦点位置を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**FocusPosition** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**FocusPosition** [= *Smallint*]

【設定値】
焦点位置

デフォルトは 0 です。

【解説】
エプソン製スキャナ用ドライバご利用時に有効です。焦点位置(原稿にピントが合うセンサの位置)を調整できます。
焦点位置は 0.1mm 単位で設定します。たとえば、1.0mm の場合は 10 を設定します。
UiMode プロパティが vikScanNONUI の場合に有効です。
データソースがサポートしている焦点位置の範囲や値は、**GetCapRange** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

FpuException(イメージキットコントロール／Scan プロパティ)

【機能】
FPU 例外をすべて無効にするかどうかを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**FpuException** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Scan.FpuException** [= *Boolean*]

【設定値】

値	説明
True	すべて無効
False	有効

【解説】
データソースや TWAIN ドライバのバージョンによっては取り込み時に浮動小数点演算エラーが発生する場合があります。その場合、このプロパティを True に設定することでエラーを回避することができます。
デフォルトは False です。

【値の設定】 実行時

【値の参照】 実行時

Gamma(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のガンマを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Gamma** [= *float*]
(2)Delphi　*imagekitcontrolname*.**Scan.Gamma** [= *Single*]

【設定値】
デフォルトは 2.2。

【解説】
UiMode プロパティが vikScanNONUI の場合に有効です。
ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
設定可能な値については、**GetCapRange** メソッドもしくは **GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

Halftone(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のハーフトーンを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Halftone** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**Halftone** [= *string*]

【設定値】
ハーフトーン名。

【解説】
UiMode プロパティが vikScanNONUI、**PixelType** プロパティが vikScanPixelBW、**BitDepthReduction** プロパティが vikBitDepthReductionHalftone の場合に有効です。
ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
設定可能な値については、**GetCapEnumToString** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
プロパティの名称が **HalfTone** から変更されました。

Highlight,Shadow,Threshold(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のハイライト・シャドウ・しきい値を設定します。

【書式】
※Highlight にて説明(Shadow,Threshold も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->Scan->Highlight [= *short*]
(2)Delphi　*imagekitcontrolname*.Scan.Highlight [= *Smallint*]

【設定値】
0～255。

Highlight はハイライト。(デフォルトは 255)
Shadow はシャドウ。(デフォルトは 0)
Threshold はしきい値。(デフォルトは 128)

【解説】
UiMode プロパティが vikScanNONUI の場合に有効です。
Highlight,Shadow プロパティは、PixelType プロパティが vikScanPixelBW 以外の場合に有効です。
Threshold プロパティは、PixelType プロパティが vikScanPixelBW、BitDepthReduction プロパティが vikBitDepthReductionThreshold の場合に有効ですが、しきい値の機能を明るさで表すデータソースの場合は Threshold プロパティではなく Brightness プロパティをご使用ください。
上記のプロパティは、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
設定可能な値については、GetCapRange メソッドもしくは GetCapEnumToFloat メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

IgnoreBackColor(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の白黒/カラー自動判別時に原稿の背景色を無視するかどうかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**IgnoreBackColor** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Scan.IgnoreBackColor** [= *Boolean*]

【設定値】

値	説明
True	背景色を無視する
False	背景色を無視しない(デフォルト)

【解説】
パナソニック製スキャナ用ドライバご利用時に有効です。
UiMode プロパティが vikScanNONUI、**PixelType** プロパティが vikScanPixelAutomation の場合に有効です。

【値の設定】 実行時

【値の参照】 不可

ImageFilter(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のイメージフィルタを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**ImageFilter** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.ImageFilter** [= *Smallint*]

【設定値】

値	説明
0	なし
1	自動
2	LOWPASS
3	BANDPASS
4	HIGHPASS

定数(vikImageFilterNone = 0, vikImageFilterAuto = 1, vikImageFilterLowPass = 2, vikImageFilterBandPass = 3, vikImageFilterHiPass = 4, vikImageFilterText = 3, vikImageFilterFineLine = 4)を使用することも可能です。

【解説】
画像の品質を改善します。LOWPASS はハーフトーン画像、BANDPASS はテキストが含まれる画像、HIGHPASS は線が含まれる画像に適しています。
UiMode プロパティが vikScanNONUI の場合に有効です。だたし、デジタルカメラからの取り込みの場合は無効です。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
・定数を使用する場合、識別子の先頭に v が付加されました(ActiveX は ikImageFilterNone, ikImageFilterAuto, ikImageFilterLowPass, ikImageFilterBandPass, ikImageFilterHiPass, ikImageFilterText, ikImageFilterFineLine)。
・プロパティの型が 4 バイト型ではなく 2 バイト型です。

ImageMerge(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の両面合成モードを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**ImageMerge** [= *TVIkScanImageMerge*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**ImageMerge** [= *TVIkScanImageMerge*]

【TVIkScanImageMerge 型】
ユニット
ImageKit
type
TVIkScanImageMerge = (vikScanImageMergeNone = 0, vikScanImageMergeFrontOnTop = 1,
vikScanImageMergeFrontOnBottom = 2, vikScanImageMergeFrontOnLeft = 3, vikScanImageMergeFrontOnRight = 4);

【設定値】

値	説明
vikScanImageMergeNone	なし(合成しない)
vikScanImageMergeFrontOnTop	表面が上で裏面が下になる(上下の貼り合わせ)
vikScanImageMergeFrontOnBottom	裏面が上で表面が下になる(上下の貼り合わせ)
vikScanImageMergeFrontOnLeft	表面が左で裏面が右になる(左右の貼り合わせ)
vikScanImageMergeFrontOnRight	裏面が左で表面が右になる(左右の貼り合わせ)

【解説】
ADF 両面取り込み時に vikScanImageMergeNone 以外を設定すると表面と裏面を合成して、一つの画像データにします。
両面合成機能を使用して得られる出力イメージは、回転を組み合わせることによっても変わります。
デフォルトは vikScanImageMergeNone です。**UiMode** プロパティが vikScanNONUI、**ScanMode** プロパティが vikScanADF2 の場合に有効です。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

【ImageKit9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikScanImageMergeNone, ikScanImageMergeFrontOnTop, ikScanImageMergeFrontOnBottom, ikScanImageMergeFrontOnLeft, ikScanImageMergeFrontOnRight)。

Indicator(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のインジケータの有無を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Indicator** [= *bool*]
(2)Delphi　*imagekitcontrolname*.**Scan.Indicator** [= *Boolean*]

【設定値】

値	説明
True	インジケータ表示(デフォルト)
False	インジケータ非表示

【解説】
UiMode プロパティが vikScanNONUI の場合に有効です。
ただし、データソースが機能をサポートしていない場合は無効です。

【値の設定】 実行時

【値の参照】 実行時

InformationFileName(イメージキットコントロール／Scan プロパティ)

【機能】
メーカ提供 UI の設定情報を保存するファイル名を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**InformationFileName** [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.**Scan.InformationFileName** [= *string*]

【設定値】
保存するファイル名。

【解説】
メーカ提供 UI を表示して設定した情報をファルに保存すると、そのファイルを開発者作成 UI にて使用することができます。一度メーカ提供 UI を表示する必要はありますが、設定情報を保存すると、次回から開発者作成 UI にて取り込みが可能です。

例:

```
//メーカ提供 UI を表示して設定情報をファイルに保存
Ret: Boolean;
VImageKit1.Scan.UiMode := vikScanSetUIOnly;
VImageKit1.Scan.InformationFileName := 'ui.dat';
Ret = VImageKit1.Scan.Exec();

//情報ファイルを使用して取り込み
Ret: Boolean;
VImageKit1.Scan.UiMode := vikScanNONUI;
VImageKit1.Scan.ScanMode := vikInformationFile;
VImageKit1.Scan.InformationFileName := 'ui.dat';
Ret := VImageKit1.Scan.Exec();
```

【値の設定】 実行時

【値の参照】 不可

JpegQuality(イメージキットコントロール／Scan プロパティ)

【機能】
ファイルおよびメモリ転送時における圧縮方法が JPEG の場合の品質係数を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**JpegQuality** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.JpegQuality** [= *Smallint*]

【設定値】
-4:未定義、-3:低品質、-2:中品質、-1:高品質、0～100。

【解説】
TransferMode プロパティが vikScanFile もしくは vikScanMemory で **Compression** プロパティが vikScanJPEG の場合に有効です。設定可能な値については、**GetCapRange** メソッドもしくは **GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

Manufacturer,ProductFamily,ProductName,ProtocolMajor,ProtocolMinor,SourceMajor, SourceMinor,SourceVersionInfo(イメージキットコントロール／Scan プロパティ)

【機能】
Select メソッドや GetDataSourceInfo メソッドで取得されるデータソースの情報を示します。

【書式】
※Manufacturer にて説明(ProductFamily,ProductName,SourceVersionInfo も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->Scan->Manufacturer [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.Scan.Manufacturer [= *string*]

※ProtocolMajor にて説明(ProtocolMinor,SourceMajor,SourceMinor も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->Scan->ProtocolMajor [= *short*]
(2)Delphi　*imagekitcontrolname*.Scan.ProtocolMajor [= *Smallint*]

【参照値】
データソースの情報。
Manufacturer は製造者名。
ProductFamily は製品ファミリー名。
ProductName は製品名。
ProtocolMajor はサポートする TWAIN のメジャーバージョン番号。
ProtocolMinor はサポートする TWAIN のマイナーバージョン番号。
SourceMajor はデータソースのメジャーバージョン番号。
SourceMinor はデータソースのマイナーバージョン番号。
SourceVersionInfo はデータソースのバージョン情報。

【値の設定】 不可

【値の参照】 実行時

MoireFilter(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のモアレ除去フィルタのレベルを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**MoireFilter** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.MoireFilter** [= *Smallint*]

【設定値】

値	説明
エプソン製スキャナ用ドライバ	
0	なし
1	標準(汎用的な設定)
2	85lpi(新聞など)
3	133lpi(週刊誌やカタログなど)
4	175lpi(写真など)
キヤノン製 DR スキャナ用ドライバ	
1	なし
2	高速
3	高画質
PFU 製スキャナ用ドライバ	
0	なし
1	モアレ除去レベル 1
2	モアレ除去レベル 2
3	モアレ除去レベル 3
4	モアレ除去レベル 4
パナソニック製スキャナ用ドライバ	
0	なし
1	あり

【解説】
エプソン製スキャナ用ドライバ、キヤノン製DRスキャナ用ドライバ、PFU製スキャナ用ドライバ、パナソニック製スキャナ用ドライバご利用時に有効です。エプソン製スキャナ用ドライバご利用時に段階設定が可能なのはエプソンスキャナドライバ『EPSON Scan』ご利用時となります。
キヤノン製 DR スキャナ用ドライバでは解像度が 300dpi 以下の場合に有効です。PFU 製スキャナ用ドライバでは **Sharpness** プロパティが 0 の場合に有効です。パナソニック製スキャナ用ドライバではモアレ除去とマルチストリームを同時に使用することはできません(どちらか一方を無効にしなければなりません)。

UiMode プロパティが vikScanNONUI の場合に有効です。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。なお、エプソン製スキャナ用ドライバのバージョンによっては有効無効の 2 つしか設定できないものがありますが、ImageKit ではその場合 0 か 0 以外で判定します。
データソースが機能をサポートしていない場合は無効です。

【値の設定】 実行時

【値の参照】 不可

NoiseFilter(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のノイズ除去フィルタを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**NoiseFilter** [= *TVIkNoiseFilter*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**NoiseFilter** [= *TVIkNoiseFilter*]

【TVIkNoiseFilter 型】
ユニット
ImageKit
type
TVIkNoiseFilter = (vikNoiseFilterNone, vikNoiseFilterAuto, vikNoiseFilterLonePixel, vikNoiseFilterMajorityRule);

【設定値】

値	説明
vikNoiseFilterNone	なし
vikNoiseFilterAuto	自動
vikNoiseFilterLonePixel	LONEPIXEL
vikNoiseFilterMajorityRule	MAJORITYRULE

【解説】
白黒 2 値画像からノイズを除去します。**UiMode** プロパティが vikScanNONUI の場合に有効です。
だたし、デジタルカメラからの取り込みの場合は無効です。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikNoiseFilterNone, ikNoiseFilterAuto, ikNoiseFilterLonePixel, ikNoiseFilterMajorityRule)。

Orientation(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の用紙の向き、または画像の回転角度を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Orientation** [= *TVIkScanOrientation*]
(2)Delphi　*imagekitcontrolname*.**Scan.Orientation** [= *TVIkScanOrientation*]

【TVIkScanOrientation 型】
ユニット
IkInit
type
TVIkScanOrientation = (vikScanRotate0 = 0, vikScanRotate90 = 1, vikScanRotate180 = 2, vikScanRotate270 = 3, vikScanTextOrientationRecognition = 1000);

【設定値】

値	説明
vikScanRotate0	0 度(ポートレイト)
vikScanRotate90	90 度
vikScanRotate180	180 度
vikScanRotate270	270 度(ランドスケープ)
vikScanTextOrientationRecognition	文字向き検知または自動回転

【解説】
デフォルトは vikScanRotate0 です。用紙の向きは 0 度を基準に時計回りに回転します。
UiMode プロパティが vikScanNONUI の場合に有効です。
原稿台や ADF に置いた用紙の出力結果を回転させたり、原稿台や ADF に用紙を横向きにセットする場合などに使用します。
ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。

vikScanTextOrientationRecognition は読み取った画像の中にある文字の向きを認識し、文字の向きが正常になるように画像を 90 度単位で回転させます。キヤノン製 DR スキャナ用ドライバでは「文字向き検知」、パナソニック製スキャナ用ドライバでは「原稿方向補正」という表現を用いています。
また、TWAIN 仕様書に記載されている自動回転に対応しているドライバについても vikScanTextOrientationRecognition にて動作いたします。
vikScanTextOrientationRecognition がサポートされているかどうかについては **IsCapSupported** メソッドで判定できます。
vikScanTextOrientationRecognition をサポートしているデータソースで vikScanTextOrientationRecognition を指定した出力結果が適切でない場合は用紙を縦向きにセットしてお試しください。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました。
ImageKit7 ActiveX は ikScanRotate0, ikScanRotate90, ikScanRotate180, ikScanRotate270 で、ImageKit8 ActiveX は vikScanTextOrientationRecognition が追加されました。

【ImageKit7 ActiveX/VCL との違い】
文字向き検知または自動回転の設定が可能となりました。

OverScan(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のオーバースキャンの取り扱いを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**OverScan** [= *TVIkOverScan*]
(2)Delphi　*imagekitcontrolname*.**Scan.OverScan** [= *TVIkOverScan*]

【TVIkOverScan 型】
ユニット
ImageKit
type
TVIkOverScan = (vikOverScanNone, vikOverScanAuto, vikOverScanTopBottom, vikOverScanLeftRight, vikOverScanAll);

【設定値】

値	説明
vikOverScanNone	指定されたサイズで読み取りを行います。
vikOverScanAuto	指定されたサイズを考慮して自動で読み取りを行います。
vikOverScanTopBottom	指定されたサイズを基に上下を広げて読み取りを行います。
vikOverScanLeftRight	指定されたサイズを基に左右を広げて読み取りを行います。
vikOverScanAll	指定されたサイズを基に上下左右を広げて読み取りを行います。

【解説】
傾いた用紙を取り込む場合などに取り込み領域(**RectLeft**,**RectTop**,**RectRight**,**RectBottom**,**PaperSize** プロパティ)を広げることができます。
UiMode プロパティが vikScanNONUI の場合に有効です。ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
データソースがサポートしている値は **GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

【ImageKit8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikScanNone, ikScanAuto, ikScanTopBottom, ikScanLeftRight, ikScanAll)。

PageCount(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のページ数を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**PageCount** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**PageCount** [= *Smallint*]

【設定値】

値	説明
0 以下	全ての原稿またはイメージを取り込む(ADF、デジタルカメラ)
1 以上	指定した数の原稿またはイメージを取り込む

【解説】
ADF にある全てのページ、およびデジタルカメラ内の全ての画像を取り込む場合は 0 以下を指定します。デジタルカメラからの取り込みで配列を使用する場合は、その要素数を指定します。**ScanMode** プロパティが vikScanADF2 の場合は奇数を指定しても取り込まれるページ数は偶数になります(両面取り込みのため)。
UiMode プロパティが vikScanNONUI 以外の場合や原稿台からの取り込みの場合は無効です。

PixelType プロパティが vikScanPixelGrayAndBW,vikScanPixelRGBAndBW の場合は 1 ページにつき 2 つのイメージが作成されるため、偶数を指定してください。たとえば、1 ページの原稿を両面で vikScanPixelGrayAndBW,vikScanPixelRGBAndBW で取り込む場合は 0 以下または 4 を指定します。なお、パナソニック製スキャナ用ドライバご利用時に vikScanPixelGrayAndBW,vikScanPixelRGBAndBW で取り込む場合は常に 0 以下として取り扱います(1 以上は無視されます)。

(注意)
データソースが機能をサポートしていない場合や正常に機能しない場合は、**PageCount** 分取り込みを行った後で処理を中断いたしますので、機種によっては ADF から用紙が取り込まれた状態のまま、用紙が内部に残る場合があります。

【値の設定】 実行時

【値の参照】 不可

PaperSize(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の用紙サイズを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**PaperSize** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.PaperSize** [= *Smallint*]

【設定値】

値	説明	サイズ
0	ユーザ定義サイズ	
1	A4	210mm x 297mm
2	JIS B5	182mm x 257mm
3	US レター	8.5″ x 11.0″ (216mm x 280mm)
4	US リーガル	11.0″ x 17.0″ (216mm x 356mm)
5	A5	148mm x 210mm
6	ISO B4	250mm x 353mm
7	ISO B6	125mm x 176mm
9	US レジャー	11.0″ x 17.0″ (280mm x 432mm)
10	US エグゼクティブ	7.25″ x 10.5″ (184mm x 267mm)
11	A3	297mm x 420mm
12	ISO B3	353mm x 500mm
13	A6	105mm x 148mm
14	C4	229mm x 324mm
15	C5	162mm x 229mm
16	C6	114mm x 162mm
17	4A0	1682mm x 2378mm
18	2A0	1189mm x 1682mm
19	A0	841mm x 1189mm
20	A1	594mm x 841mm
21	A2	420mm x 594mm
22	A7	74mm x 105mm
23	A8	52mm x 74mm
24	A9	37mm x 52mm
25	A10	26mm x 37mm
26	ISO B0	1000mm x 1414mm
27	ISO B1	707mm x 1000mm
28	ISO B2	500mm x 707mm
29	ISO B5	176mm x 250mm
30	ISO B7	88mm x 125mm
31	ISO B8	62mm x 88mm
32	ISO B9	44mm x 62mm
33	ISO B10	31mm x 44mm
34	JIS B0	1030mm x 1456mm
35	JIS B1	728mm x 1030mm
36	JIS B2	515mm x 728mm
37	JIS B3	364mm x 515mm
38	JIS B4	257mm x 364mm
39	JIS B6	128mm x 182mm
40	JIS B7	91mm x 128mm
41	JIS B8	64mm x 91mm
42	JIS B9	45mm x 64mm
43	JIS B10	32mm x 45mm
44	C0	917mm x 1297mm
45	C1	648mm x 917mm

46	C2	458mm x 648mm
47	C3	324mm x 458mm
48	C7	81mm x 114mm
49	C8	57mm x 81mm
50	C9	40mm x 57mm
51	C10	28mm x 40mm
52	US ステイトメント	5.5″ x 8.5″ (140mm x 216mm)
53	ビジネスカード	90mm x 55mm
54	最大サイズ	
1000	未定義サイズ	

定数(vikScanUserSize = 0, vikScanA4LETTER = 1, vikScanB5LETTER = 2, vikScanUSLETTER = 3, vikScanUSLEGAL = 4, vikScanA5 = 5, vikScanB4 = 6, vikScanB6 = 7, vikScanUSLEDGER = 9, vikScanUSEXECUTIVE = 10, vikScanA3 = 11, vikScanB3 = 12, vikScanA6 = 13, vikScanC4 = 14, vikScanC5 = 15, vikScanC6 = 16, vikScan4A0 = 17, vikScan2A0 = 18, vikScanA0 = 19, vikScanA1 = 20, vikScanA2 = 21, vikScanA4 = 1, vikScanA7 = 22, vikScanA8 = 23, vikScanA9 = 24, vikScanA10 = 25, vikScanISOB0 = 26, vikScanISOB1 = 27, vikScanISOB2 = 28, vikScanISOB3 = 12, vikScanISOB4 = 6, vikScanISOB5 = 29, vikScanISOB6 = 7, vikScanISOB7 = 30, vikScanISOB8 = 31, vikScanISOB9 = 32, vikScanISOB10 = 33, vikScanJISB0 = 34, vikScanJISB1 = 35, vikScanJISB2 = 36, vikScanJISB3 = 37, vikScanJISB4 = 38, vikScanJISB5 = 2, vikScanJISB6 = 39, vikScanJISB7 = 40, vikScanJISB8 = 41, vikScanJISB9 = 42, vikScanJISB10 = 43, vikScanC0 = 44, vikScanC1 = 45, vikScanC2 = 46, vikScanC3 = 47, vikScanC7 = 48, vikScanC8 = 49, vikScanC9 = 50, vikScanC10 = 51, vikScanUSSTATEMENT = 52, vikScanBUSINESSCARD = 53, vikScanMaxSize = 54, vikScanUndefinedSize = 1000)を使用することも可能です。

【解説】

UiMode プロパティが vikScanNONUI の場合に有効です。ただし、デジタルカメラからの取り込みは無効です。
RectLeft,**RectTop**,**RectRight**,**RectBottom** プロパティを有効にする場合は 0 以下を指定します。
また、データソースによってはユーザ定義サイズ(0)と読み取り範囲を設定しても、開始位置などが正しく反映されない場合があります。その場合は、定型サイズ(1 以上)を符号反転してお試しください。(下記の例を参照)

例:

```
(PaperSize = 0 で読み取り範囲を設定 --> 読み取り位置がおかしい)
VImageKit1.Scan.UiMode := vikScanNONUI;
VImageKit1.Scan.UnitMode := vikScanCM;
VImageKit1.Scan.PaperSize := 0;
VImageKit1.Scan.RectLeft := 0;
VImageKit1.Scan.RectTop := 0;
VImageKit1.Scan.RectRight := 6.9;
VImageKit1.Scan.RectBottom := 10.2;

(PaperSize に定型サイズを符号反転して読み取り範囲を設定 --> 正しく読み取れる)
VImageKit1.Scan.UiMode := vikScanNONUI;
VImageKit1.Scan.UnitMode := vikScanCM;
VImageKit1.Scan.PaperSize := -3; // US レター
VImageKit1.Scan.RectLeft := 0;
VImageKit1.Scan.RectTop := 0;
VImageKit1.Scan.RectRight := 6.9;
VImageKit1.Scan.RectBottom := 10.2;
```

PaperSizeプロパティにはデータソースがサポートしている用紙サイズ、もしくはその中で一番大きな用紙サイズを設定してください。なお、データソースによっては効果がない場合もありますので、詳しくはスキャナメーカにご確認ください。

(注意)

データソースがサポートしている用紙サイズは **GetCapEnumToFloat** メソッドで取得できます。データソースによっては定型サイズをサポートしていないものや、用紙サイズの機能自体をサポートしていないものがありますので、その場合は 0 を指定してください。
未定義サイズ(1000)を設定すると、用紙サイズを自動検知して取り込みを行いますが、用紙の幅と高さのどちらか一方のみを検知するスキャナでは用紙の自動検知が正しく行われない場合があります。
未定義サイズがサポートされているかどうかについては **IsCapSupported** メソッドで判定できます。
未定義サイズがサポートされていて **PaperSize** プロパティに 1000 を設定しても用紙サイズの自動検知が正しく動作しない場

合は次の手順をお試しください。
(1)**BorderDetection** プロパティに true を設定する。この段階で正しく動作せずに PFU 製スキャナ用ドライバをご利用の場合は(2)へ。
(2)**PaperSize** プロパティに 1000 に加えて読み取り可能な最大サイズの用紙サイズを設定する。たとえば、最大サイズが A3 であれば PaperSize = 1000 + 11 を設定する。

【値の設定】 実行時

【値の参照】 不可

【ImageKit7/8/9/10 ActiveX との違い】
・定数を使用する場合、識別子の先頭に v が付加されました(ActiveX は ikScanUserSize, ikScanA4LETTER, ikScanB5LETTER, ikScanUSLETTER, ikScanUSLEGAL, ikScanA5, ikScanB4, ikScanB6, ikScanUSLEDGER, ikScanUSEXECUTIVE, ikScanA3, ikScanB3, ikScanA6, ikScanC4, ikScanC5, ikScanC6, ikScan4A0, ikScan2A0, ikScanA0, ikScanA1, ikScanA2, ikScanA4, ikScanA7, ikScanA8, ikScanA9, ikScanA10, ikScanISOB0, ikScanISOB1, ikScanISOB2, ikScanISOB3, ikScanISOB4, ikScanISOB5, ikScanISOB6, ikScanISOB7, ikScanISOB8, ikScanISOB9, ikScanISOB10, ikScanJISB0, ikScanJISB1, ikScanJISB2, ikScanJISB3, ikScanJISB4, ikScanJISB5, ikScanJISB6, ikScanJISB7, ikScanJISB8, ikScanJISB9, ikScanJISB10, ikScanC0, ikScanC1, ikScanC2, ikScanC3, ikScanC7, ikScanC8, ikScanC9, ikScanC10, ikScanUSSTATEMENT, ikScanBUSINESSCARD, ikScanMaxSize, ikScanUndefinedSize)。
・プロパティの型が 4 バイト型ではなく 2 バイト型です。

PixelType(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のイメージの種類を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**PixelType** [= *TVIkScanPixelType*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**PixeltType** [= *TVIkScanPixelType*]

【TVIkScanPixelType 型】
ユニット
IkInit
type
TVIkScanPixelType = (vikScanPixelBW = 0, vikScanPixelGray = 1, vikScanPixelRGB = 2, vikScanPixelPalette = 3, vikScanPixelCMY = 4, vikScanPixelCMYK = 5, vikScanPixelYUV = 6, vikScanPixelYUVK = 7, vikScanPixelCIEXYZ = 8, vikScanPixelAutomation = 1000, vikScanPixelGrayAndBW = 1001, vikScanPixelRGBAndBW = 1002);

【設定値】

値	説明
vikScanPixelBW	白黒 2 値(1 ビット)
vikScanPixelGray	グレースケール(4 ビット、8 ビット、12 ビット、14 ビット、16 ビット)
vikScanPixelRGB	RGB カラー(24 ビット、36 ビット、42 ビット、48 ビット)
vikScanPixelPalette	パレットカラー(4 ビット、8 ビット)
vikScanPixelAutomation	白黒/カラー自動判別(1 ビット or 24 ビット)
vikScanPixelGrayAndBW	白黒 2 値とグレースケール(1 ビットと 8 ビット)
vikScanPixelRGBAndBW	白黒 2 値と RGB カラー(1 ビットと 24 ビット)

イメージキットコントロールでは vikScanPixelCMY, vikScanPixelCMYK, vikScanPixelYUV, vikScanPixelYUVK, vikScanPixelCIEXYZ はサポートしておりません。

【解説】
デフォルトは vikScanPixelRGB です。**UiMode** プロパティが vikScanNONUI の場合に有効です。
ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
データソースがサポートしているピクセルタイプは、**GetCapEnumToFloat** メソッドで取得できます。

vikScanPixelAutomationはパナソニック製スキャナ用ドライバ(メモリ転送時は非圧縮形式のみ)、キヤノン製DRスキャナ用ドライバ、エプソン製スキャナ用ドライバ、PFU製スキャナ用ドライバご利用時に有効です。
vikScanPixelGrayAndBW,vikScanPixelRGBAndBWはキヤノン製DRスキャナ用ドライバ、パナソニック製スキャナ用ドライバ(メモリ転送時は非圧縮形式のみ)とPFU製スキャナ用ドライバ、エプソン製スキャナ用ドライバご利用時に有効です。
vikScanPixelAutomation,vikScanPixelGrayAndBW,vikScanPixelRGBAndBW がサポートされているかどうかについては **IsCapSupported** メソッドで判定できます。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました。
ImageKit7 ActiveX は ikScanPixelBW, ikScanPixelGray, ikScanPixelRGB, ikScanPixelPalette, ikScanPixelAutomation で、
ImageKit8 ActiveX は vikScanPixelGrayAndBW, vikScanPixelRGBAndBW が追加されました。

【ImageKit7 ActiveX/VCL との違い】
マルチストリームでの取り込みが可能となりました。

【ImageKit8 ActiveX/VCL との違い】
エプソン製スキャナ用ドライバのダブルイメージ出力(マルチストリーム)に対応しました。

RangeCurrent,RangeDefault,RangeMax,RangeMin,RangeStep
(イメージキットコントロール／Scan プロパティ)

【機能】
GetCapRange メソッドで取得される情報を示します。

【書式】
※RangeCurrent にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->Scan->RangeCurrent [= *float*]
(2)Delphi　*imagekitcontrolname*.Scan.RangeCurrent [= *Single*]

【参照値】
RangeCurrent は現在設定されている値。
RangeDefault はデフォルト値。(デバイスの電源を入れた時の値)
RangeMax は最大値。
RangeMin は最小値。
RangeStep はステップ値。(範囲内[RangeMin～RangeMax]の隣接する値との差)

【解説】
GetCapRange メソッドで指定した項目(X 方向解像度など)の情報を取得できます。

【値の設定】 不可

【値の参照】 実行時

RectBottom,RectLeft,RectRight,RectTop（イメージキットコントロール／Scan プロパティ）

【機能】
開発者作成 UI で取り込む時の取込位置を取得または設定します。

【書式】
※RectBottom にて説明（RectLeft,RectRight,RectTop も同様な使い方）
(1)C++Builder　*imagekitcontrolname*->Scan->RectBottom [= *float*]
(2)Delphi　*imagekitcontrolname*.Scan.RectBottom [= *Single*]

【設定値】
UnitMode プロパティに応じた値で設定します（UnitMode プロパティが vikScanInch の場合は、インチ単位）。
最小値は 0。

【解説】
RectBottom,RectLeft,RectRight,RectTop の各プロパティには、それぞれ取込む原稿の下、左、右、上の位置を設定します。
UiMode プロパティが vikScanNONUI 以外の場合や PaperSize プロパティが 1 以上およびデジタルカメラからの取り込みの場合、設定値は無効です。
取り込み終了後に UnitMode プロパティの単位に応じた値が設定されます。
データソースのデフォルト値を使用する場合は RectLeft,RectTop,RectRight,RectBottom に 0 を設定します。

【値の設定】 実行時

【値の参照】 実行時

RemoveHole(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のパンチ穴の取り扱いを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**RemoveHole** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.RemoveHole** [= *Smallint*]

【設定値】

値	説明
0:	パンチ穴を除去しない
1:	パンチ穴を除去する(白で埋める)
2:	パンチ穴を除去する(周辺色で埋める)

デフォルトは 0 です。

【解説】
パンチ穴を除去します。キヤノン製DRスキャナ用ドライバ、エプソン製スキャナ用ドライバ、パナソニック製スキャナ用ドライバ、PFU製スキャナ用ドライバご利用時に有効です。
"白で埋める"と"周辺色で埋める"の機能は PFU 製スキャナ用ドライバご利用時のみ有効です。

UiMode プロパティが vikScanNONUI の場合に有効です。ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
データソースが機能をサポートしているかどうかについては、**IsCapSupported** メソッドで判定できます。

【値の設定】 実行時

【値の参照】 不可

RotateBack(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の ADF を使用した両面スキャン時の裏面回転を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**RotateBack** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.RotateBack** [= *Smallint*]

【設定値】

値	説明
0	裏面を回転しない(デフォルト)
1	裏面を 180 度回転する

【解説】
キヤノン製DRスキャナ用ドライバおよびエプソン製スキャナ用ドライバ『EPSON Scan』ご利用時に有効ですが、『EPSON Scan』のVer.1.xにおいてはA4 サイズを超える原稿では無効です。
プロパティ値が 1 の場合、キヤノン製 DR スキャナ用ドライバでは UI を表示した時の「裏面を+180 度回転する」のオプションを有効にした場合と同等です。エプソン製スキャナ用ドライバでは『EPSON Scan』の環境設定において「裏面の向きを表面に合わせる」オプションを有効にした場合と同等です。
UiMode プロパティが vikScanNONUI、**ScanMode** プロパティが vikScanADF2 の場合に有効です。
ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
データソースが機能をサポートしているかどうかについては、**IsCapSupported** メソッドで判定できます。

【値の設定】 実行時

【値の参照】 不可

Rotation(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の画像の回転角度を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Rotation** [= *int*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**Rotation** [= *Integer*]

【設定値】
-360～360(単位は度)。
デフォルトは 0 です。

【解説】
0 度を基準に負の値の場合は反時計回り、正の値の場合は時計回りに回転します。**UiMode** プロパティが vikScanNONUI の場合に有効です。**Orientation** プロパティと機能が似ていますが、**Orientation** プロパティへの設定可能な値が vikScanRotate0 のみの場合は **Rotaion** プロパティを使用してください。Orientation と Rotaion の機能を両方ともサポートするデータソースの場合、双方の値によって回転角度が決まったり、**Rotaion** プロパティの値が有効になったりします。そのため、**Rotaion** プロパティに 0 以外を設定する場合は **Orientation** プロパティに vikScanRotate0 を設定するといいでしょう。
ただし、**Orientation** プロパティに vikScanTextOrientationRecognition が設定されている場合やデジタルカメラからの取り込み、およびデータソースが機能をサポートしていない場合は無効です。
データソースがサポートしている値は **GetCapEnumToFloat** メソッドもしくは **GetCapRange** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

ScanCount(イメージキットコントロール／Scan プロパティ)

【機能】
取込んだイメージの数を示します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**ScanCount** [= *int*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**ScanCount** [= *Integer*]

【参照値】
取り込んだイメージの数。

【解説】
Execute メソッド実行後に、値が **ScanCount** プロパティに設定されます。

【値の設定】 不可

【値の参照】 実行時

ScanMode(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の取り込みモードを取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**ScanMode** [= *TVIkScanMode*]
(2)Delphi　*imagekitcontrolname*.**Scan.ScanMode** [= *TVIkScanMode*]

【TVIkScanMode 型】
ユニット
IkInit
type
TVIkScanMode = (vikScanDOC = 0, vikScanADF1 = 1, vikScanADF2 = 2, vikScanCAM = 3, vikScanCAMArray = 4, vikScanCAMThumb = 5, vikScanCAMThumbArray = 6, vikPositiveFilm1 = 7, vikPositiveFilm2 = 8, vikInformationFile = 100);

【設定値】

値	説明
vikScanDOC	反射原稿 － 原稿台
vikScanADF1	反射原稿 － ADF 片面(デジタルカメラのオリジナル画像)
vikScanADF2	反射原稿 － ADF 両面
vikScanCAM	デジタルカメラのオリジナル画像
vikScanCAMArray	デジタルカメラのオリジナル画像(配列使用)
vikScanCAMThumb	デジタルカメラのサムネイル画像
vikScanCAMThumbArray	デジタルカメラのサムネイル画像(配列使用)
vikPositiveFilm1	透過原稿 － ポジフィルム、フィルムホルダ使用
vikPositiveFilm2	透過原稿 － ポジフィルム、フィルムエリアガイド使用
vikInformationFile	メーカ提供 UI の設定情報ファイルを使用

【解説】
デフォルトは vikScanDOC です。**UiMode** プロパティが vikScanNONUI の場合に有効です。
ScanMode プロパティが vikScanCAMArray,vikScanCAMThumbArray の場合は、**Execute** メソッドの引数で与える配列が有効となりますが、それ以外は無効です。ImageKit5 ActiveX との互換性を保つため、**ScanMode** プロパティを vikScanADF1、**XResolution** プロパティを 0 に設定した場合はデジタルカメラからの取り込みとなります(オリジナル画像)。

vikPositiveFilm1,vikPositiveFilm2 の場合はエプソン製スキャナ用ドライバご利用時に有効です。vikPositiveFilm2 はフィルムエリアガイドをサポートする機種で使用可能です。
vikInformationFile の場合は **InformationFileName** プロパティにメーカ提供 UI の設定情報が保存されたファイル名を設定します。

(注意)
両面読み取り機能をサポートしているスキャナでも、ユーザインタフェースを非表示にすると正しく読み取りできない場合があります。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました。
ImageKit7 ActiveX は ikScanDOC, ikScanADF1, ikScanADF2, ikScanCAM, ikScanCAMArray, ikScanCAMThumb, ikScanCAMThumbArray で、ImageKit8 ActiveX は ikPositiveFilm1, ikPositiveFilm2, ikInformationFile が追加されました。

【ImageKit7 ActiveX/VCL との違い】
エプソン製スキャナの透過原稿(ポジフィルム)とメーカ提供 UI の設定情報による取り込みが可能となりました。

ScanningSpeed(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の読み取り速度を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**ScanningSpeed** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.ScanningSpeed** [= *Smallint*]

【設定値】

値	説明
エプソン製スキャナ用ドライバ	
0	品質優先
1	速度優先
パナソニック製スキャナ用ドライバ	
0	低速
1	標準
2	高速

【解説】
エプソン製スキャナ用ドライバおよびパナソニック製スキャナ用ドライバご利用時に有効です。
デフォルトは 0 です。**UiMode** プロパティが vikScanNONUI の場合に有効です。
読み取り速度を速くすると遅い場合に比べて画質が粗くなりますので、用途に応じてご利用ください。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

Sharpness(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のシャープネスのレベルを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**Sharpness** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.Sharpness** [= *Smallint*]

【設定値】

値	説明

キヤノン製 DR スキャナ用ドライバ(エッジ強調)

0～5	0 の場合はドライバのデフォルト値 3 と同じ動作となります。 1 の場合は画像の輪郭が柔らかくなり、5 の場合は画像の輪郭がくっきりします。

エプソン製スキャナ用ドライバ(アンシャープマスク)

0	なし
1	弱
2	中
3	強

パナソニック製スキャナ用ドライバ(画質)

0	なし
1	スムーズ
2	低
3	標準
4	高
5	自動

PFU 製スキャナ用ドライバ(輪郭強調)

0	なし
1	弱
2	中
3	強

【解説】
キヤノン製DRスキャナ用ドライバ、エプソン製スキャナ用ドライバ、パナソニック製スキャナ用ドライバ、PFU製スキャナ用ドライバご利用時に有効です。エプソン製スキャナ用ドライバご利用時に段階設定が可能なのはエプソンスキャナドライバ『EPSON Scan』ご利用時となります。
キヤノン製 DR スキャナ用ドライバでは「エッジ強調」、エプソン製スキャナ用ドライバでは「アンシャープマスク」、パナソニック製スキャナ用ドライバでは「画質」、PFU 製スキャナ用ドライバでは「輪郭強調」という表現を用いています。

デフォルトは 0 です。**UiMode** プロパティが vikScanNONUI の場合に有効です。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。なお、エプソン製スキャナ用ドライバのバージョンによっては有効無効の 2 つしか設定できないものがありますが、ImageKit ではその場合 0 か 0 以外で判定します。

【値の設定】 実行時

【値の参照】 不可

SkipBlankPage(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の白紙ページの取り扱いを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**SkipBlankPage** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.SkipBlankPage** [= *Smallint*]

【設定値】

値	説明
-2	白紙ページを除去しない
-1	白紙ページを除去する

キヤノン製 DR スキャナ用ドライバ

0	白紙ページを除去しない
1	白紙ページを除去する

エプソン製スキャナ用ドライバ

0	白紙ページを除去しない
1	白紙ページを除去する(レベル低)
2	白紙ページを除去する(レベル中)
3	白紙ページを除去する(レベル高)

【解説】
白紙ページを除去します。キヤノン製DRスキャナ用ドライバとパナソニック製スキャナ用ドライバでは**SkipBlankThreshold**プロパティも併せて設定します。
キヤノン製DRスキャナ用ドライバで白紙ページを除去する場合は-1 または 1 を設定してください。また、**ScanMode**プロパティを必ずvikScanADF1 に設定してください。白紙ページ除去が有効の場合は**ScanMode**プロパティがvikScanADF1 であっても両面取り込みとなります。
エプソン製スキャナ用ドライバで白紙ページを除去する場合は-1 または 1 以上を設定してください。白紙ページの判別レベルを設定する場合は 1 以上とし、ガンマ補正(AdjustGamma)がサポートされている場合は**AdjustGamma**プロパティを 1 に設定してください。**AdjustGamma**プロパティが 0 の場合、白紙ページ除去機能が正常に動作しない可能性があります。また、ガンマ補正(AdjustGamma)がサポートされていない場合は、デフォルトの設定でスキャンを行うと暗めの画像が取得され、白紙ページ除去機能が正常に動作しない可能性があります。その場合は**Brightness**プロパティや**Gamma**プロパティの値を調整して明るめの画像にしてください。
UiMode プロパティが vikScanNONUI の場合に有効です。ただし、デジタルカメラからの取り込みやデータソースが機能をサポートしていない場合は無効です。
データソースがサポートしている値は **GetCapEnumToFloat** メソッドもしくは **GetCapRange** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 不可

SkipBlankThreshold(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の白紙ページに含まれるしきい値の割合や白紙スキップのしやすさを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan->SkipBlankThreshold** [= *float*]
(2)Delphi　*imagekitcontrolname*.**Scan.SkipBlankThreshold** [= *Single*]

【設定値】
キヤノン製 DR スキャナ用ドライバ
0.1 から 20.0 までの値を%単位で設定します。

パナソニック製スキャナ用ドライバ
0.01 から 5.00 までの値を%単位で設定します。

デフォルトは 1 です。

【解説】
キヤノン製DRスキャナ用ドライバとパナソニック製スキャナ用ドライバご利用時に有効です。キヤノン製DRスキャナ用ドライバでは白紙スキップのしやすさを表し、値が大きければ白紙ページとして認識される確率が高くなります。パナソニック製スキャナ用ドライバでは白紙ページに含まれるしきい値の割合を表し、値が小ければ白紙ページとして認識される確率が高くなります。
UiMode プロパティが vikScanNONUI、**SkipBlankPage** プロパティが-1 または 1 の場合に有効です。

【値の設定】 実行時

【値の参照】 不可

TextEnhancement(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のテキストエンハンスメントの種類を設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**TextEnhancement** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.TextEnhancement** [= *Smallint*]

【設定値】

値	説明
キヤノン製 DR スキャナ用ドライバ	
0	適用しない
1	テキストエンハンスメント
2	アドバンスドテキストエンハンスメント
3	高速テキストエンハンスメント
4,5	アドバンスドテキストエンハンスメントⅡ(ご利用の機種により値が異なる)
エプソン製スキャナ用ドライバ	
0	適用しない
1	文字くっきり(標準)
2	文字くっきり(強)

【解説】
キヤノン製DRスキャナ用ドライバ、エプソン製スキャナ用ドライバご利用時に有効です。エプソン製スキャナ用ドライバご利用時に段階設定が可能なのはエプソンスキャナドライバ『EPSON Scan Ver.5.0 以降』ご利用時となります。
キヤノン製 DR スキャナ用ドライバでは **PixelType** プロパティが vikScanPixelBW で **BitDepthReduction** プロパティが vikBitDepthReductionThreshold の場合に有効です。
キヤノン製 DR スキャナ用ドライバでは「テキストエンハンスメント」、エプソン製スキャナ用ドライバでは「文字くっきり」という表現を用いています。

デフォルトは 0 です。**UiMode** プロパティが vikScanNONUI の場合に有効です。
データソースがサポートしている値は、**GetCapEnumToFloat** メソッドで取得できます。

なお、エプソン製スキャナ用ドライバのバージョンによっては有効無効の 2 つしか設定できないものがありますが、ImageKit ではその場合 0 か 0 以外で判定します。データソースが機能をサポートしていない場合は無効です。

【値の設定】 実行時

【値の参照】 不可

TransferMode(イメージキットコントロール／Scan プロパティ)

【機能】
イメージの転送方法を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**TransferMode** [= *TVIkScanTransfer*]
(2)Delphi　*imagekitcontrolname*.**Scan.TransferMode** [= *TVIkScanTransfer*]

【TVIkScanTransfer 型】
ユニット
IkInit
type
TVIkScanTransfer = (vikScanNative, vikScanFile, vikScanMemory, vikScanFile2);

【設定値】

値	説明
vikScanNative	ネイティブ(DIB で転送)
vikScanFile	ファイル
vikScanMemory	メモリ(ブロック単位で転送)
vikScanFile2	ファイル 2 *1

*1 ImageKit10 ではサポートしておりません。

【解説】
デフォルトは vikScanNative です。**Execute** メソッドを実行する前に設定する必要があります。
データソースがサポートしている転送方法は、**GetCapEnumToFloat** メソッドで取得できます。
Scanning イベントが発生するのはメモリ転送の場合のみです。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikScanNative, ikScanFile, ikScanMemory, ikScanFile2)。

UiMode(イメージキットコントロール／Scan プロパティ)

【機能】
取り込む時のユーザインタフェースの状態を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**UiMode** [= *TVIkScanUIMode*]
(2)Delphi　*imagekitcontrolname*.**Scan.UiMode** [= *TVIkScanUIMode*]

【TVIkScanUIMode 型】
ユニット
IkInit
type
TVIkScanUIMode = (vikScanUI, vikScanUICLOSE, vikScanNONUI, vikScanSetUIOnly);

【設定値】

値	説明
vikScanUI	メーカ提供 UI 表示－取り込み後閉じない
vikScanUICLOSE	メーカ提供 UI 表示－取り込み後閉じる
vikScanNONUI	メーカ提供 UI 非表示(開発者作成 UI)
vikScanSetUIOnly	メーカ提供 UI 表示－値の設定のみで取り込み不可

【解説】
デフォルトは vikScanUI です。**Execute** メソッドを実行する前に設定する必要があります。
メーカ提供 UI 非表示での取り込みやメーカ提供 UI を表示して値の設定のみできる機能をサポートしているかどうかについては、**IsCapSupported** メソッドで判定できます。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました。
ImageKit7 ActiveX は ikScanUI, ikScanUICLOSE, ikScanNONUI で、ImageKit8 ActiveX は ikScanSetUIOnly が追加されました。

【ImageKit7 ActiveX/VCL との違い】
メーカ提供 UIを表示して設定情報を保存することが可能となりました(**InformationFileName**プロパティに保存するファイル名を設定します)。

UnitFlag(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の取り込み単位をデータソースの機能に任せるか、ImageKit10 で考慮するかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**UnitFlag** [= *short*]
(2)Delphi　*imagekitcontrolname*.**Scan.UnitFlag** [= *Smallint*]

【設定値】

値	説明
0	データソースの機能を使用
1	ImageKit10 で考慮

【解説】
デフォルトは 0 です。**UiMode** プロパティが vikScanNONUI の場合に有効です。
例えば、データソースがサポートしていない単位を使用したい場合や、**PaperSize** プロパティが 0 以下で **RectLeft**,**RectTop**, **RectRight**,**RectBottom** プロパティを設定しているのに正しく取り込みができない、または取り込み範囲が有効にならない場合などは 1 を指定してください。
XResolution、**YResolution** プロパティが共に-1 で **UnitMode** プロパティが vikScanPixel の場合は無効です。

【値の設定】 実行時

【値の参照】 不可

UnitMode(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時の取り込み単位を取得または設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->**Scan**->**UnitMode** [= *TVIkScanUnit*]
(2)Delphi　*imagekitcontrolname*.**Scan**.**UnitMode** [= *TVIkScanUnit*]

【TVIkScanUnit 型】
ユニット
IkInit
type
TVIkScanUnit = (vikScanInch, vikScanCM, vikScanPica, vikScanPoint, vikScanTwip, vikScanPixel, vikScanMillimeter);

【設定値】

値	説明
vikScanInch	インチ
vikScanCM	cm(センチメートル)
vikScanPica	パイカ
vikScanPoint	ポイント
vikScanTwip	twip
vikScanPixel	ピクセル
vikScanMillimeter	mm(ミリメートル)

【解説】
デフォルトは vikScanInch です。**UiMode** プロパティが vikScanNONUI の場合に有効となり、
RectLeft,**RectTop**,**RectRight**,**RectBottom** プロパティに設定する値の単位となります。
UnitFlag プロパティが 0 の場合は、データソースがサポートしている値でなければなりません。
UiMode プロパティが vikScanNONUI 以外の場合、設定値は無効ですが、取り込み時に使用した単位を返します。データソースがサポートしている取り込み単位は、**GetCapEnumToFloat** メソッドで取得できます。

【値の設定】 実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました。
ImageKit7 ActiveX は ikScanInch, ikScanCM, ikScanPica, ikScanPoint, ikScanTwip, ikScanPixel で、ImageKit8 ActiveX は vikScanMillimeters が追加されました。
識別子の中に含まれる Millimeters が Millimeter に変更されました。

【ImageKit8 VCL との違い】
vikScanMillimeters が vikScanMillimeter に変更されました。

XResolution,YResolution(イメージキットコントロール／Scan プロパティ)

【機能】
イメージの読取解像度を取得または設定します。

【書式】
※XResolution にて説明(YResolution も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->Scan->XResolution [= *int*]
(2)Delphi　*imagekitcontrolname*.Scan.XResolution [= *Integer*]

【設定値】
X 方向および Y 方向の読取り解像度を DPI 単位で設定。
XResolution は X 方向の解像度。
YResolution は Y 方向の解像度。

【解説】
UiMode プロパティが vikScanNONUI の場合に設定した解像度が有効になります。
UiMode プロパティが vikScanNONUI 以外の場合やデジタルカメラからの取り込みの場合は設定値は無効です。データソースがサポートしている解像度の範囲や値は、GetCapEnumToFloat メソッドもしくは GetCapRange メソッドで取得できます。
データソースのデフォルト値を使用する場合は-1 を設定します。

(注意)
X 方向と Y 方向の読み取り解像度には異なる値を設定できますが、データソースによっては異なる解像度をサポートしていないものがあります。その場合、X 方向および Y 方向のどちらかの解像度が有効になります。実際の読み取り解像度は BeforeScan イベント内もしくは Execute メソッド終了後にプロパティを参照してください。

【値の設定】 実行時

【値の参照】 実行時

XScaling,YScaling(イメージキットコントロール／Scan プロパティ)

【機能】
開発者作成 UI で取り込む時のイメージのスケーリングの比率を設定します。

【書式】
※**XScaling** にて説明(**YScaling** も同様な使い方)
(1)C++Builder　　*imagekitcontrolname*->**Scan**->**XScaling** [= *float*]
(2)Delphi　　*imagekitcontrolname*.**Scan**.**XScaling** [= *Single*]

【設定値】
X 方向および Y 方向のスケーリングの比率を設定。(デフォルトは 1.0)
XScaling は X 方向の比率。
YScaling は Y 方向の比率。

【解説】
UiMode プロパティが vikScanNONUI の場合に有効です。
UiMode プロパティが vikScanNONUI 以外の場合やデジタルカメラからの取り込みの場合は無効です。
データソースがサポートしているスケーリングの比率の範囲や値は、**GetCapEnumToFloat** メソッドもしくは **GetCapRange** メソッドで取得できます。
等倍(実寸)で読み取る場合は 1.0 を設定します。

(注意)
X 方向とY 方向のスケーリングの比率には異なる値を設定できますが、データソースによっては異なるスケーリングをサポートしていないものがあります。その場合、X 方向および Y 方向のどちらかのスケーリングが有効になります。

【値の設定】 実行時

【値の参照】 不可

ClearProperty(イメージキットコントロール／Scan メソッド)

【機能】
イメージキットコントロールの Scan プロパティを初期化します。

【書式】
(1)C++Builder　*imagekitcontrolname*->Scan->ClearProperty()
(2)Delphi　*imagekitcontrolname*.Scan.ClearProperty

【引数】
ありません。

【戻り値】
ありません。

【解説】
値を設定可能なプロパティを初期化します。参照専用プロパティは除きます。

CloseDataSource,OpenDataSource（イメージキットコントロール／Scan メソッド）

【機能】
CloseDataSource: データソースをクローズします。
OpenDataSource: データソースをオープンします。

【書式】
※CloseDataSource にて説明（OpenDataSource も同様な使い方）
(1)C++Builder [*bool =*]*imagekitcontrolname*->Scan->CloseDataSource()
(2)Delphi [*Boolean =*]*imagekitcontrolname*.Scan.CloseDataSource

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
CloseDataSource:
OpenDataSource メソッドでオープンしたデータソースをクローズします。

OpenDataSource:
DataSourceName プロパティに設定されたデータソースをオープンします。
DataSourceName プロパティに空文字列を指定する場合は、予め Select メソッドを実行しておいてください。
（""を設定した場合、Select メソッドで選択されたデータソースとなります。）
OpenDataSource メソッドを実行すると、CloseDataSource メソッドを実行するまでの間、オープンしたデータソースを占有することができ、他のアプリケーションからは利用できなくなります。

CloseDataSource,OpenDataSource メソッドを実行するには、事前に Initialize メソッドが実行されていなければなりません。
CloseDataSource,OpenDataSource メソッドは以前の ImageKit で提供していたスキャンコントロールの ScanCloseDS,ScanOpenDS メソッドに相当します。

【ImageKit7/8/9/10 ActiveX との違い】
メソッドの名称が CloseDS,OpenDS からそれぞれ変更されました。

Execute(イメージキットコントロール／Scan メソッド)

【機能】
TWAIN データソースからイメージの取込を実行します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Scan**->**Execute**(const int * Index, const int Index_Size)
[*bool* =]*imagekitcontrolname*->**Scan**->**Execute**()
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Scan**.**Execute**(const Index: array of Integer)
[*Boolean* =]*imagekitcontrolname*.**Scan**.**Execute**

【引数】

名称	内容
Index	デジタルカメラから取り込む際に使用するインデックス配列

※C++Builder の場合、Index の要素数-1 を Index_Size に与えます。
※インデックス配列を使用しない場合は引数なしのメソッドをご利用ください。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
イメージを 1 枚取り込む前後および最中にイベント(**BeforeScan**,**AfterScan**,**Scanning**)が発生しますので、その中で処理したいコードを記述してください。
Index は **UiMode** プロパティが vikScanNONUI で **ScanMode** プロパティがデジタルカメラを表す場合に有効となりますが、Index が不要な場合は引数なしのメソッドをご利用ください。
Index は、デジタルカメラから指定した画像を取り込む際に使用します。(**ScanMode** プロパティが vikScanCAMArray, vikScanCAMThumbArray の時に有効)
例えば、5 番目の画像と 8 番目の画像を取り込むには
```
Index[0] := 5;
Index[1] := 8;
```
とします。
データソースによってはサポートされていない機能があります。その場合はサポートされている機能のみを使用して取り込みを行います。

Execute メソッドを実行するには、事前に **Initialize** メソッドが実行されていなければなりません。
Execute メソッドを実行する前に **OpenDataSource** メソッドを実行すると、オープン中のデータソースは他のアプリケーションからは利用できなくなり、データソースを占有することができます。

Execute メソッドは以前の ImageKit で提供していたスキャンコントロールの **ScanExec** メソッドに相当します。

＜メソッドを実行する前に設定する必要のあるプロパティ＞
「必須項目」
DataSourceName(空文字列を指定する場合は、予め **Select** メソッドを実行しておくこと)
※**OpenDataSource** メソッドを事前に実行した場合は **DataSourceName** は無効(オープンされたデータソースが有効)
TransferMode,**Compression**,**UiMode**(エプソン製スキャナ用ドライバご利用時は **ExtUiMode** も必須)
「設定条件により必要な項目」
FileFormat,**FileName**,**InformationFileName**,**JpegQuality**

UiModeがvikScanNONUIでScanModeがvikInformationFileの場合
ScanMode,**PageCount**,**UnitMode**

UiMode が 2(ikScanNONUI)で ScanMode が vikInformationFile 以外の場合
(1)スキャナ
AdjustGamma,**AutoBright**,**BitDepth**,**BitDepthReduction**,**Border**,**BorderColor**,**BorderDetection**,**Brightness**,**ColorBWRati**

o,Contrast,Deskew,DropoutColor,DynamicThreshold,FocusPosition,Gamma,Halftone,Highlight,IgnoreBackColor,ImageFilter,ImageMerge,Indicator,MoireFilter,NoiseFilter,Orientation,OverScan,PageCount,PaperSize,PixelType,RemoveHole,Rotation,ScanMode,ScanningSpeed,Shadow,Sharpness,SkipBlankPage,SkipBlankThreshold,TextEnhancement,Threshold,UnitFlag,UnitMode,XResolution,XScaling,YResolution,YScaling
PaperSize<=0: RectBottom,RectLeft,RectRight,RectTop
※
AdjustGamma,Border,BorderColor,ColorBWRatio,DynamicThreshold,FocusPosition,IgnoreBackColor,MoireFilter,RemoveHole,RotateBack,ScanningSpeed,Sharpness,SkipBlankThreshold,TextEnhancement はご利用のドライバによっては無効です。

(2)デジタルカメラ
PageCount,ScanMode,UnitMode　　※ScanMode = vikScanADF1 の場合は、XResolution = 0 とすること。

＜メソッド実行後に取得されるプロパティ＞
BitDepth,Compression,ErrorStatus,FileFormat(ファイル転送の場合),PixelType,RectBottom,RectLeft,RectRight,RectTop,ScanCount,UnitMode,XResolution,YResolution

設定するプロパティに関しては、それぞれのプロパティの解説を参照してください。

(開発者作成 UI を使用する上での注意事項)
UiMode が vikScanNONUI の場合、機種やプロパティの設定状態により読み取りに失敗する場合があります。エラーが発生する主な原因を次に示しますので参考にしてください。

・設定した取り込み単位がサポートされていない。サポートされている値は GetCapEnumToFloat メソッドで取得できます。
・設定したピクセルタイプがサポートされていない。サポートされている値は GetCapEnumToFloat メソッドで取得できます。
・設定した画素ビット数がサポートされていない。サポートされている値は GetBitDepth メソッドで取得できます。
・設定した解像度がサポートされていない。サポートされている値は GetCapEnumToFloat,GetCapRange メソッドで取得できます。
・設定した用紙サイズがサポートされていない。サポートされている値は GetCapEnumToFloat メソッドで取得できます。
(1)用紙サイズそのものがサポートされていない場合
PaperSize プロパティに 0(ユーザ定義サイズ)を設定し、RectLeft,RectTop,RectRight,RectBottom プロパティに範囲内の値を設定する。読み取り範囲は、GetMinimumSize と GetPhysicalSize メソッドで取得できます。
(2)設定した用紙サイズがサポートされているのに読み取りに失敗する場合
PaperSize プロパティに 0(ユーザ定義サイズ)以外を設定してご確認ください。

コード例:
Delphi
```
var
  Ret: Boolean;
begin
  Ret := VImageKit1.Scan.Initialize(1, 0, '1.00', 'NEWTONE Corp. ', 'ImageKit', 'ImageKit Scan Sample');
  if Ret = False Then Exit;

  VImageKit1.Scan.ClearProperty;
  VImageKit1.Scan.TransferMode := vikScanMemory;
  VImageKit1.Scan.Compression := vikScanNoCompress;
  VImageKit1.Scan.UiMode := vikScanUI;
  VImageKit1.Scan.DataSourceName := 'EPSON TWAIN Pro';
  Ret := VImageKit1.Scan.Execute;

  Ret := VImageKit1.Scan.Terminate;
end;

//イベント
{$IFDEF CONDITIONALEXPRESSIONS}
  {$IF CompilerVersion >= 23.0}
procedure TForm1.VImageKit1AfterScan(Sender: TObject; DibHandle,
  OrgHandle: NativeUInt; ImageCount: Integer; BitOrder: Smallint);
  {$ELSE}
```

```
    procedure TForm1.VImageKit1AfterScan(Sender: TObject; DibHandle,
      OrgHandle: Cardinal; ImageCount: Integer; BitOrder: Smallint);
      {$IFEND}
    {$ENDIF}
    begin
      VImageKit1.DisplayMode := vikScale
      VImageKit1.ImageHandle := VImageKit1.CopyImage(DibHandle);
    end;
```

【ImageKit7/8/9/10 ActiveX との違い】

・メソッドの名称が **Exec** から変更されました。
・Index に Const パラメータが付加されました。
・Delphi で使用する場合、引数の Index が配列型に変更されました。
・C++Builder で使用する場合、引数として Index_Size が追加されました。
・**ScanMode** プロパティが vikScanCAMArray, vikScanCAMThumbArray の場合に Index 配列の要素数を **PageCount** プロパティに設定する必要がなくなりました。ただし、C++Builder では引数の Index_Size に要素数を渡す必要があります。

FreeTwain（イメージキットコントロール／Scan メソッド）

【機能】

LoadTwain メソッドでロードした TWAIN DLL を解放します。

【書式】

(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Scan**->**FreeTwain**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Scan.FreeTwain**

【引数】

ありません。

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

LoadTwain メソッドを実行していない場合は **FreeTwain** メソッドを実行する必要はありません（ImageKit 内で自動的にロード/解放が行われます）。**LoadTwain/FreeTwain** メソッドは対で使用してください。
FreeTwain メソッドを実行する場合は、事前に **Terminate** メソッドを実行するようにしてください。

GetBitDepth(イメージキットコントロール／Scan メソッド)

【機能】
データソースがサポートしている画素ビット数を取得します。

【書式】
(1)C++Builder
[*int =*]*imagekitcontrolname*->**Scan**->**GetBitDepth**(short * List, const int List_Size)
(2)Delphi
[*Integer =*]*imagekitcontrolname*.**Scan**.**GetBitDepth**(var List: array of Smallint)

【引数】

名称	内容
List	画素ビット数を取得する配列

※C++Builder の場合は、配列の要素数-1 を List_Size に与えます。

【戻り値】
List と ListNum を設定した場合
取得した数を返します。0 の場合はエラーとなりますのでエラーステータスを確認してください。
List が NULL の場合
取得する際に必要な配列の要素数を返します。データソースによってはピクセルタイプの値に関係なく、サポートしている全ての画素ビット数を取得するものがあります。その場合は、実際の数よりも多い値が返されます。

【解説】
画素ビット数を取得するには、**PixelType**,**DataSourceName** プロパティを設定する必要があります。
DataSourceName プロパティに空文字列を指定する場合は、予め **Select** メソッドを実行しておいてください。
(**OpenDataSource** メソッドを事前に実行した場合は **DataSourceName** プロパティは無効で、オープンされたデータソースが有効)
PixelType プロパティにはデータソースがサポートしている値を設定してください。
スキャナによっては転送方法の値により設定可能な画素ビット数が異なる場合があります。そのため、**TransferMode** プロパティを事前に設定しておくとよいでしょう。

GetBitDepth メソッドを実行するには、事前に **Initialize** メソッドが実行されていなければなりません。
GetBitDepth メソッドを実行する前に **OpenDataSource** メソッドを実行すると、オープン中のデータソースは他のアプリケーションからは利用できなくなり、データソースを占有することができます。

GetBitDepth メソッドは以前の ImageKit で提供していたスキャンコントロールの **ScanGetBitDepth** メソッドに相当します。

配列に必要な要素数を取得してから、画素ビット数を取得する例:
(1)C++Builder

```
int Num;
short *List;

Num = VImageKit1->Scan->GetBitDepth(NULL, 0);
if (Num == 0) return;
List = new short[Num];
VImageKit1->Scan->GetBitDepth(List, Num - 1);
//様々な処理
delete[] List;
```

(2)Delphi

```
Num: Integer;
List: array of Smallint;

Num := VImageKit1.Scan.GetBitDepth(PSmallint(nil)^);
if Num = 0 then Exit;
SetLength(List, Num);
```

```
VImageKit1.Scan.GetBitDepth(List);
//様々な処理
```

【ImageKit7/8/9/10 ActiveX との違い】
Delphi で使用する場合、引数の List が配列型に変更され、配列の要素数を引数として渡す必要がなくなりました。

GetCapEnumToFloat(イメージキットコントロール／Scan メソッド)

【機能】
データソースから指定した項目の設定可能値を取得します。

【書式】
(1)C++Builder
[*int =*]*imagekitcontrolname*->**Scan**->**GetCapEnumToFloat**(Word CapNo, short &ConType, float * List, const int List_Size, int &CurrentIndex, int &DefaultIndex)
(2)Delphi
[*Integer =*]*imagekitcontrolname*.**Scan**.**GetCapEnumToFloat**(CapNo: Word; var ConType: Smallint; var List: array of Single; var CurrentIndex, DefaultIndex: Integer)

【引数】

名称	内容
CapNo	取得する項目番号(16 進表記) 0x0100　圧縮方法 0x0101　ピクセルタイプ 0x0102　取り込み単位 0x0103　転送方法 0x1100　明るさの自動化(*1) 0x1101　明るさ 0x1103　コントラスト 0x1108　ガンマ 0x110a　ハイライト 0x110c　ファイルフォーマット 0x110e　ドロップアウトカラー 0x1110　用紙の向き 0x1113　シャドウ 0x1118　X 方向解像度 0x1119　Y 方向解像度 0x1121　回転角度 0x1122　用紙サイズ 0x1123　しきい値 0x1124　X 方向スケーリング 0x1125　Y 方向スケーリング 0x112c　色深度の減少方法 0x1147　イメージフィルタ 0x1148　ノイズフィルタ 0x1149　オーバースキャン 0x1153　JPEG 品質係数 0x115c　両面合成 0xf001　読み取り速度 0xf002　モアレフィルタ 0xf004　シャープネス 0xf006　拡張 UI モード 0xf007　ダイナミックスレッシュホールド 0xf00b　白紙ページ除去 0xf00e　テキストエンハンスメント ※Delphi は 0x を$に置き換えてください。

定数(vikScanCompression = 0x0100, vikScanPixelType = 0x0101, vikScanUnits = 0x0102, vikScanTransferMode = 0x0103, vikScanAutoBright = 0x1100, vikScanBrightness = 0x1101, vikScanContrast = 0x1103, vikScanGamma = 0x1108, vikScanHighlight = 0x110a, vikFileScanFormat = 0x110c, vikScanDropoutColor = 0x110e, vikScanOrientation = 0x1110, vikScanShadow = 0x1113, vikScanXResolution = 0x1118, vikScanYResolution = 0x1119, vikScanRotation = 0x1121, vikScanPaperSize = 0x1122, vikScanThreshold = 0x1123, vikScanXScaling = 0x1124, vikScanYScaling = 0x1125,

vikScanBitDepthReduction = 0x112c, vikScanImageFilter = 0x1147, vikScanNoiseFilter = 0x1148, vikScanOverScan = 0x1149, vikScanJpegQuality = 0x1153, vikScanImageMerge = 0x115c, vikScanScanningSpeed = 0xf001, vikScanMoirefilter = 0xf002, vikScanSharpness = 0xf004, vikScanExtUiMode = 0xf006, vikScanDynamicThreshold = 0xf007, vikScanSkipBlankPage = 0xf00b, vikScanTextEnhancement = 0xf00e)を使用することも可能です。

ConType　　項目の型を取得する変数

List　　項目値を取得する配列
※C++Builder の場合は、配列の要素数-1 を List_Size に与えます。

CurrentIndex　　現在値を示す List へのインデックス(インデックスは 0 から始まります)
List[CurrentIndex] = 現在値

DefaultIndex　　デフォルト値を示す List へのインデックス(インデックスは 0 から始まります)
List[DefaultIndex] = デフォルト値

【戻り値】
ConType=3 もしくは 4 の場合は取得した数を返します。
ConType=5 の場合は 1、ConType=6 の場合は 5 を返します。(固定)
List が NULL の場合は、取得する際に必要な配列の要素数を返します。
0 の場合はエラーとなりますので、エラーステータスを確認してください。

【解説】
項目の設定可能値を取得するには **DataSourceName** プロパティを設定する必要があります。
DataSourceName プロパティに空文字列を指定する場合は、予め **Select** メソッドを実行しておいてください。
(**OpenDataSource** メソッドを事前に実行した場合は **DataSourceName** プロパティは無効で、オープンされたデータソースが有効)
スキャナによってはピクセルタイプの値(白黒 2 値や 24 ビットカラー)により設定可能な解像度が異なる場合があります。また、転送方法の値(メモリやファイル)により設定可能な圧縮方法が異なる場合があります。そのため、X 方向解像度と Y 方向解像度の設定可能値を取得する場合は、**PixelType** プロパティを、圧縮方法の設定可能値を取得する場合は、**TransferMode** プロパティを事前に設定しておくとよいでしょう。

GetCapEnumToFloat メソッドを実行するには、事前に **Initialize** メソッドが実行されていなければなりません。
GetCapEnumToFloat メソッドを実行する前に **OpenDataSource** メソッドを実行すると、オープン中のデータソースは他のアプリケーションからは利用できなくなり、データソースを占有することができます。

GetCapEnumToFloat メソッドは以前の ImageKit で提供していたスキャンコントロールの **ScanGetCapEnum** メソッドに相当します。

ConType に設定される値について
(0:メソッドの戻り値が 0 の場合、3:配列型、4:列挙型、5:単一型、6:範囲型)
ConType=3 の場合、List 配列に値が設定され、CurrentIndex と DefaultIndex は無効となります。
ConType=4 の場合、List 配列に値が設定されます。
ConType=5 の場合、List[0]に現在値が設定されます。
ConType=6 の場合、List[0]に最小値が、List[1]に最大値が、List[2]にステップ値が、List[3]にデフォルト値が、List[4]に現在値がそれぞれ設定されます。

List に設定される値の内容
CapNo が圧縮方法を表す場合:
0:非圧縮、1:PACKBITS、2:GROUP3-1D、3:GROUP3-1DEOL、4:GROUP3-2D、5:GROUP4、6:JPEG
7:LZW、8:JBIG、9:PNG、10:RLE4、11: RLE8、12:BITFIELDS
(2,3,4,5 は FAX[CCITT]、10,11,12 は BMP 形式)
CapNo がピクセルタイプを表す場合:
0:白黒 2 値、1:グレースケール、2:RGB カラー、3:パレットカラー
(データソースによっては 4～8 が設定される場合もあります)
CapNo が取り込み単位を表す場合:
0:インチ、1:cm、2:パイカ、3:ポイント、4:twip、5:ピクセル、6:mm
CapNo が転送方法を表す場合:
0:ネイティブ、1:ファイル、2:メモリ、3:ファイル 2

CapNo がファイルフォーマットを表す場合:
0:TIFF、1:PICT、2:BMP、3:XBM、4:JFIF(JPEG)、5:FPX、6:TIFF MULTI、7:PNG、8:SPIFF、9:EXIF、10:PDF
11:JP2(JPEG2000 Part1)、13:JPX(JPEG2000 Part2)、14:DEJAVU、15:PDF/A(Version 1)、16:PDF/A(Version 2)

CapNo がドロップアウトカラーを表す場合:
0:赤、1:緑、2:青、3:なし、4:白
(データソースによっては 5,6 が設定される場合もあります)
CapNo が用紙の向きを表す場合:
0:0 度(ポートレイト)、1:90 度、2:180 度、3:270 度(ランドスケープ)
CapNo が用紙サイズを表す場合:
0:ユーザ定義サイズ、1:A4、2:JIS B5、3:US レター、4:US リーガル、5:A5、6:ISO B4、7:ISO B6、9:US レジャー
10:US エグゼクティブ、11:A3、12:ISO B3、13:A6、14:C4、15:C5、16:C6、17:4A0、18:2A0、19:A0、20:A1、21:A2
22:A7、23:A8、24:A9、25:A10、26:ISO B0、27:ISO B1、28:ISO B2、29:ISO B5、30:ISO B7、31:ISO B8
32:ISO B9、33:ISO B10、34:JIS B0、35:JIS B1、36:JIS B2、37:JIS B3、38:JIS B4、39:JIS B6、40:JIS B7、41:JIS B8
42:JIS B9、43:JIS B10、44:C0、45:C1、46:C2、47:C3、48:C7、49:C8、50:C9、51:C10、52:US ステイトメント
53:ビジネスカード、54:最大サイズ
CapNo が色深度の減少方法を表す場合:
0:しきい値、1:ハーフトーン、2:カスタムハーフトーン、3:拡散、4:動的しきい値
CapNo がイメージフィルタを表す場合:
0:なし、1:自動、2:LOWPASS、3:BANDPASS、4:HIGHPASS
CapNo がノイズフィルタを表す場合:
0:なし、1:自動、2:LONEPIXEL、3:MAJORITYRULE
CapNo がオーバースキャンを表す場合:
0:指定されたサイズで読み取り、1:指定されたサイズを考慮して自動で読み取り
2:指定されたサイズを基に上下を広げて読み取り、3:指定されたサイズを基に左右を広げて読み取り
4:指定されたサイズを基に上下左右を広げて読み取り
CapNo が両面合成を表す場合:
0:なし(合成しない)、1:指定されたサイズを考慮して自動で読み取り
2:裏面が上で表面が下になる(上下の貼り合わせ)、3:表面が左で裏面が右になる(左右の貼り合わせ)
4:裏面が左で表面が右になる(左右の貼り合わせ)
CapNo が読み取り速度を表す場合:
エプソン製スキャナ用ドライバおよびパナソニック製スキャナ用ドライバご利用時に有効です。
(1)エプソン製スキャナ用ドライバ　0:品質優先、1:速度優先
(2)パナソニック製スキャナ用ドライバ　0:低速、1:通常、2:高速
CapNo がモアレフィルタを表す場合:
エプソン製スキャナ用ドライバ、キヤノン製 DR スキャナ用ドライバ、PFU 製スキャナ用ドライバ、パナソニック製スキャナ用ドライバご利用時に有効です。
(1)エプソン製スキャナ用ドライバ
0:なし、1:標準、2:85lpi(新聞など)、3:133lpi(週刊誌やカタログなど)、4:175lpi(写真など)
(2)キヤノン製 DR スキャナ用ドライバ
1:なし、2:高速、3:高画質
(3)PFU 製スキャナ用ドライバ
0:なし、1:モアレ除去レベル 1、2:モアレ除去レベル 2、3:モアレ除去レベル 3、4:モアレ除去レベル 4
(4)パナソニック製スキャナ用ドライバ
0:なし、1:あり
CapNo がシャープネスを表す場合:
キヤノン製 DR スキャナ用ドライバ、エプソン製スキャナ用ドライバ、パナソニック製スキャナ用ドライバ、PFU 製スキャナ用ドライバご利用時に有効です。
キヤノン製 DR スキャナ用ドライバでは「エッジ強調」、エプソン製スキャナ用ドライバでは「アンシャープマスク」、パナソニック製スキャナ用ドライバでは「画質」、PFU 製スキャナ用ドライバでは「輪郭強調」という表現を用いています。
(1)キヤノン製 DR スキャナ用ドライバ　1 から 5
(2)エプソン製スキャナ用ドライバ　0:なし、1:小、2:中(デフォルト)、3:大
(3)パナソニック製スキャナ用ドライバ　0:なし、1:スムーズ、2:低、3:標準、4:高、5:自動
(4)PFU 製スキャナ用ドライバ　0:なし、1:弱、2:中、3:強
CapNo が拡張 UI モードを表す場合:
エプソン製スキャナ用ドライバご利用時に有効です。
0:前回設定の UI モードで起動、1:プロフェッショナルモードで起動、2:全自動モードで起動、3:ホームモードで起動
4:全自動モードで起動(エラーメッセージを除き、UI 非表示)、5:オフィスモードで起動
CapNo がダイナミックスレッシュホールドを表す場合:

パナソニック製スキャナ用ドライバご利用時に有効です。
0:なし、1:明るい、2:少し明るい、3:標準、4:少し暗い、5:暗い
CapNo がテキストエンハンスメントを表す場合:
キヤノン製 DR スキャナ用ドライバ、エプソン製スキャナ用ドライバご利用時に有効です。
(1)キヤノン製 DR スキャナ用ドライバ
0:適用しない、1:テキストエンハンスメント、2:アドバンスドテキストエンハンスメント、3:高速テキストエンハンスメント
4,5:アドバンスドテキストエンハンスメントⅡ
(2)エプソン製スキャナ用ドライバ
0:適用しない、1:文字くっきり(標準)、2:文字くっきり(強)

CapNo が明るさの自動化、明るさ、コントラスト、ガンマ、ハイライト、シャドウ、X 方向解像度、Y 方向解像度、回転角度、しきい値、X 方向スケーリング、Y 方向スケーリング、JPEG 品質係数、白紙ページ除去を表す場合は設定可能な値となります。
ただし、ConType=6 の場合は **GetCapRange** メソッドを実行した結果と同じとなります。
(*1)データソースが TWAIN2.0 以降に対応している場合は設定可能な値となりますが、それ以外は現在値となります。

配列に必要な要素数を取得してから、取り込み単位を取得する例:
(1)C++Builder

```
int Num;
float *List;
short ConType;
int CurrentIndex, DefaultIndex;

Num = VImageKit1->Scan->GetCapEnumToFloat(0x0102, ConType, NULL, 0, CurrentIndex, DefaultIndex);
if (Num == 0) return;
List = new float[Num];
VImageKit1->Scan->GetCapEnumToFloat(0x0102, ConType, List, Num - 1, CurrentIndex, DefaultIndex);
//様々な処理
delete[] List;
```

(2)Delphi

```
Num: Integer;
List: array of Single;
ConType: Smallint;
CurrentIndex, DefaultIndex: Integer;

Num := VImageKit1.Scan.GetCapEnumToFloat($0102, ConType, PSingle(nil)^, CurrentIndex, DefaultIndex);
if Num = 0 then Exit;
SetLength(List, Num);
VImageKit1.Scan.GetCapEnumToFloat($0102, ConType, List, CurrentIndex, DefaultIndex);
//様々な処理
```

【ImageKit7/8/9/10 ActiveX との違い】
・**GetCapEnum** メソッドに相当しますが、引数の CapNo にハーフトーンを渡すことはできません。
・列挙型の識別子の先頭に v が付加されました。

【ImageKit7 ActiveX/VCL との違い】
CapNo に回転角度、オーバースキャン、白紙ページ除去、テキストエンハンスメントの値が追加されました。

【ImageKit8 ActiveX/VCL との違い】
CapNo に両面合成の値が追加されました。

【ImageKit9 ActiveX/VCL との違い】
CapNo に明るさの自動化の値が追加されました。

GetCapEnumToString(イメージキットコントロール／Scan メソッド)

【機能】
データソースから指定した項目の設定可能値を取得します。

【書式】
(1)C++Builder
[*int =*]*imagekitcontrolname*->**Scan**->**GetCapEnumToString**(Word CapNo, short &ConType, UnicodeString * List, const int List_Size, int &CurrentIndex, int &DefaultIndex)
(2)Delphi
[*Integer =*]*imagekitcontrolname*.**Scan**.**GetCapEnumToString**(CapNo: Word; var ConType: Smallint; var List: array of string; var CurrentIndex, DefaultIndex: Integer)

【引数】

名称	内容
CapNo	取得する項目番号(16 進表記) 0x1109　ハーフトーン ※Delphi は 0x を$に置き換えてください。 定数(vikScanHalftone = 0x1109)を使用することも可能です。
ConType	項目の型を取得する変数
List	項目値を取得する配列 ※C++Builder の場合は、配列の要素数-1 を List_Size に与えます。
CurrentIndex	現在値を示す List へのインデックス(インデックスは 0 から始まります) List[CurrentIndex] = 現在値
DefaultIndex	デフォルト値を示す List へのインデックス(インデックスは 0 から始まります) List[DefaultIndex] = デフォルト値

【戻り値】
取得した数を返します。
0 の場合はエラーとなりますので、エラーステータスを確認してください。

【解説】
項目の設定可能値を取得するには **DataSourceName** プロパティを設定する必要があります。
DataSourceName プロパティに空文字列を指定する場合は、予め **Select** メソッドを実行しておいてください。
(**OpenDataSource** メソッドを事前に実行した場合は **DataSourceName** プロパティは無効で、オープンされたデータソースが有効)

GetCapEnumToString メソッドを実行するには、事前に **Initialize** メソッドが実行されていなければなりません。
GetCapEnumToString メソッドを実行する前に **OpenDataSource** メソッドを実行すると、オープン中のデータソースは他のアプリケーションからは利用できなくなり、データソースを占有することができます。

GetCapEnumToString メソッドは以前の ImageKit で提供していたスキャンコントロールの **ScanGetCapEnum** メソッドに相当します。

ConType に設定される値について
(0:メソッドの戻り値が 0 の場合、3:配列型、4:列挙型、5:単一型、6:範囲型)
ConType=3 の場合、List 配列に値が設定され、CurrentIndex と DefaultIndex は無効となります。
ConType=4 の場合、List 配列に値が設定されます。
ConType=5 の場合、List[0]に現在値が設定されます。
ConType=6 の場合はありません。

List に設定される値の内容
ハーフトーンを表す名称が設定されます。

配列に必要な要素数を取得してから、値を取得する例:
(1)C++Builder

```
int Num;
UnicodeString *List;
short ConType;
int CurrentIndex, DefaultIndex;

Num = VImageKit1->Scan->GetCapEnumToString(0x1109, ConType, NULL, 0, CurrentIndex, DefaultIndex);
if (Num == 0) return;
List = new UnicodeString[Num];
VImageKit1->Scan->GetCapEnumToString(0x1109, ConType, List, Num - 1, CurrentIndex, DefaultIndex);
//様々な処理
delete[] List;
```

(2)Delphi

```
Num: Integer;
List: array of string;
ConType: Smallint;
CurrentIndex, DefaultIndex: Integer;

Num := VImageKit1.Scan.GetCapEnumToString($1109, ConType, PString(nil)^, CurrentIndex, DefaultIndex);
if Num = 0 then Exit;
SetLength(List, Num);
VImageKit1.Scan.GetCapEnumToString($1109, ConType, List, CurrentIndex, DefaultIndex);
//様々な処理
```

【ImageKit7/8/9/10 ActiveX との違い】
・**GetCapEnum** メソッドに相当しますが、**HalfToneList** プロパティに設定される値が引数の List に設定されます。
・定数の識別子の先頭に v が付加されました。

GetCapRange(イメージキットコントロール／Scan メソッド)

【機能】
データソースから指定した項目の範囲を取得します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Scan**->**GetCapRange**(Word CapNo, short &ConType)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Scan.GetCapRange**(CapNo: Word; var ConType: Smallint)

【引数】

名称	内容
CapNo	取得する項目番号(16 進表記) 0x1101　明るさ 0x1103　コントラスト 0x1108　ガンマ 0x110a　ハイライト 0x1113　シャドウ 0x1118　X 方向解像度 0x1119　Y 方向解像度 0x1121　回転角度 0x1123　しきい値 0x1124　X 方向スケーリング 0x1125　Y 方向スケーリング 0x1153　JPEG 品質係数 0xf004　シャープネス 0xf00b　白紙ページ除去 0xf00d　焦点位置 ※Delphi は 0x を$に置き換えてください。
	定数(vikScanBrightness = 0x1101, vikScanContrast = 0x1103, vikScanGamma = 0x1108, vikScanHighlight = 0x110a, vikScanShadow = 0x1113, vikScanXResolution = 0x1118, vikScanYResolution = 0x1119, vikScanRotation = 0x1121, vikScanThreshold = 0x1123, vikScanXScaling = 0x1124, vikScanYScaling = 0x1125, vikScanJpegQuality = 0x1153, vikScanSharpness = 0xf004, vikScanSkipBlankPage = 0xf00b, viikScanFocusPosition = 0xf00d)を使用することも可能です。
ConType	項目の型を取得する変数

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
項目の範囲を取得するには **DataSourceName** プロパティを設定する必要があります。
DataSourceName プロパティに空文字列を指定する場合は、予め **Select** メソッドを実行しておいてください。
(**OpenDataSource** メソッドを事前に実行した場合は **DataSourceName** プロパティは無効で、オープンされたデータソースが有効)
スキャナによってはピクセルタイプの値(白黒 2 値や 24 ビットカラー)により設定可能な解像度が異なる場合があります。そのため、X 方向解像度と Y 方向解像度の設定可能値を取得する場合は、**PixelType** プロパティを事前に設定しておくとよいでしょう。

GetCapRange メソッドを実行するには、事前に **Initialize** メソッドが実行されていなければなりません。
GetCapRange メソッドを実行する前に **OpenDataSource** メソッドを実行すると、オープン中のデータソースは他のアプリケーションからは利用できなくなり、データソースを占有することができます。
GetCapRange メソッドは以前の ImageKit で提供していたスキャンコントロールの **ScanGetCapRange** メソッドに相当します。

ConType に設定される値について

（0：メソッドの戻り値が False の場合、3:配列型、4:列挙型、5：単一型、6：範囲型）

ConType=3 または 4 の場合、**RangeMax** プロパティに項目を取得する際に必要な要素数が設定されますので、その要素数を基に **GetCapEnumToFloat** メソッドを実行してください。
ConType=5 の場合、**RangeCurrent** プロパティに現在値が、**RangeDefault** プロパティにデフォルト値が設定されます。
ConType=6 の場合、**RangeCurrent**,**RangeDefault**,**RangeMax**,**RangeMin**,**RangeStep** プロパティに値が設定されます。

【ImageKit7/8/9/10 ActiveX との違い】
定数の識別子の先頭に v が付加されました。

【ImageKit7 ActiveX/VCL との違い】
CapNo に回転角度、シャープネス、白紙ページ除去、焦点位置の値が追加されました。

GetDataSourceInfo(イメージキットコントロール／Scan メソッド)

【機能】
指定したデータソースの情報を取得します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->Scan->GetDataSourceInfo()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.Scan.GetDataSourceInfo

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
データソースの情報を取得するには DataSourceName プロパティを設定する必要があります。
DataSourceName プロパティに空文字列を指定する場合は、予め Select メソッドを実行しておいてください。
成功した場合、Manufacturer,ProductFamily,ProductName,ProtocolMajor,ProtocolMinor,SourceMajor,SourceMinor,
SourceVersionInfo プロパティにデータソースの情報が設定されます。

GetDataSourceInfo メソッドを実行するには、事前に Initialize メソッドが実行されていなければなりません。
GetDataSourceInfo メソッドは以前の ImageKit で提供していたスキャンコントロールの ScanGetDSInfo メソッドに相当します。

GetMinimumSize,GetPhysicalSize（イメージキットコントロール／Scan メソッド）

【機能】

GetMinimumSize: データソースから取り込める最小サイズを取得します。
GetPhysicalSize: データソースから取り込める最大物理サイズを取得します。

【書式】

※**GetMinimumSize** にて説明（**GetPhysicalSize** も同様な使い方）
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Scan**->**GetMinimumSize**(float &AWidth, float &AHeight)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Scan**.**GetMinimumSize**(var AWidth, AHeight: Single)

【引数】

名称	内容
AWidth	取り込める最大物理幅、もしくは最小幅を取得する変数
AHeight	取り込める最大物理高さ、もしくは最小高さを取得する変数

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

最大物理サイズもしくは最小サイズを取得するには、**DataSourceName**,**ScanMode**,**UnitMode** プロパティを設定する必要があります。
UnitMode プロパティに vikScanPixel を設定する場合は、**XResolution**,**YResolution** プロパティに解像度（DPI）を設定してください。該当する解像度に応じたサイズが取得できます。**XResolution**,**YResolution** プロパティが共に 0 の場合は、現在の解像度（DPI）に応じたサイズとなります。
DataSourceName プロパティに空文字列を指定する場合は、予め **Select** メソッドを実行しておいてください。
（**OpenDataSource** メソッドを事前に実行した場合は **DataSourceName** プロパティは無効で、オープンされたデータソースが有効）
ScanMode には vikScanDOC,vikScanADF1,vikScanADF2 のいずれか（**GetPhysicalSize** の場合は vikPositiveFilm1 と vikPositiveFilm2 も可）、**UnitMode** にはデータソースがサポートしている値を設定してください。
AWidth、AHeight には **ScanMode** および **UnitMode** に応じた値が設定されます。

GetPhysicalSize,**GetMinimumSize** メソッドを実行するには、事前に **Initialize** メソッドが実行されていなければなりません。
GetMinimumSize,**GetPhysicalSize** メソッドを実行する前に **OpenDataSource** メソッドを実行すると、オープン中のデータソースは他のアプリケーションからは利用できなくなり、データソースを占有することができます。

GetMinimumSize,**GetPhysicalSize** メソッドは以前の ImageKit で提供していたスキャンコントロールの
ScanGetMinimumSize,**ScanGetPhysicalSize** メソッドに相当します。

【ImageKit7 ActiveX/VCL との違い】

UnitMode プロパティが vikScanPixel の場合、指定した解像度に応じたサイズが取得できるようになりました。

Initialize(イメージキットコントロール／Scan メソッド)

【機能】
TWAIN の初期化を行います。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Scan**->**Initialize**(short MajorNum, short MinorNum, const UnicodeString VersionInfo, const UnicodeString Manufacturer, const UnicodeString ProductFamily, const UnicodeString ProductName)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Scan**.**Initialize**(MajorNum, MinorNum: Smallint; const VersionInfo, Manufacturer, ProductFamily, ProductName: string)

【引数】

名称	内容
MajorNum	アプリケーションのメジャーバージョン番号(バージョンが 1.00 の場合は 1)
MinorNum	アプリケーションのマイナーバージョン番号(バージョンが 1.00 の場合は 0)
VersionInfo	アプリケーションのバージョン情報(32 バイト以内)
Manufacturer	製造者名(32 バイト以内)
ProductFamily	製品ファミリー名(32 バイト以内)
ProductName	製品名(32 バイト以内)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
TWAIN による機能を使用する場合、必ず実行する必要があります。
MajorNum,MinorNum,VersionInfo,Manufacturer,ProductFamily,ProductName は TWAIN に対してアプリケーションの情報を通知するために使用されます。
Initialize メソッドは以前の ImageKit で提供していたスキャンコントロールの **ScanInitialize** メソッドに相当します。

【ImageKit7/8/9/10 ActiveX との違い】
・フォームのウィンドウハンドルを引数として渡す必要がなくなりました。
・C++Builder で使用する場合、引数の VersionInfo, Manufacturer, ProductFamily, ProductName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の VersionInfo, Manufacturer, ProductFamily, ProductName が string 型に変更されました。

IsCapSupported(イメージキットコントロール／Scan メソッド)

【機能】
データソースから指定した項目がサポートされているかどうかを確認します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Scan**->**IsCapSupported**(Word CapNo, int &Value)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Scan.IsCapSupported**(CapNo: Word; var Value: Integer)

【引数】

名称	内容
CapNo	確認する項目番号(16 進表記) 0x1002　ドキュメントフィーダー機能 0x1003　ドキュメントフィーダーに用紙があるかどうかを確認する機能 0x100b　インジケータ 0x100e　UI コントロール機能 0x100f　デバイスの稼動状態の確認機能 0x1011　サムネイル画像の転送機能 0x1012　ドキュメントフィーダーの両面機能 0x1014　設定用 UI 表示機能 0x1015　UI 設定値の取得/設定機能 0x1100　明るさの自動化 0x112d　未定義サイズ 0x112e　デジタルカメラ内の画像枚数 0x1150　自動領域切り出し機能 0x1151　デスキュー(傾き補正) 0xf003　モアレフィルタ使用時の最大有効解像度 0xf005　ADF を使用した両面スキャン時の裏面回転 0xf008　白黒/カラー自動判別読み取り機能 0xf009　マルチストリーム 0xf00a　文字向き検知または自動回転 0xf00c　パンチ穴除去 0xf00f　境界補正 0xf010　ガンマ補正 ※Delphi は 0x を$に置き換えてください。
定数(vikScanFeederEnabled = 0x1002, vikScanFeederLoaded = 0x1003, vikScanIndicator = 0x100b, vikScanUiControllable = 0x100e, vikScanDeviceonline = 0x100f, vikScanThumbnailsEnabled = 0x1011, vikScanDuplex = 0x1012, vikScanEnableDSUIOnly = 0x1014, vikScanCustomDSData = 0x1015, vikScanAutoBright = 0x1100, vikScanUndefinedImagesize = 0x112d, vikScanImageDataset = 0x112e, vikScanBorderDetection = 0x1150, vikScanDeskew = 0x1151, vikScanMoirefilterMaxResolution = 0xf003, vikScanRotateBack = 0xf005, vikScanPixelAutomationCap = 0xf008, vikScanMultiStream = 0xf009, vikScanTextOrientationRecognition = 0xf00a, vikScanRemoveHole = 0xf00c, vikScanBorder = 0xf00f, vikScanAdjustGamma = 0xf010)を使用することも可能です。	
Value	状態を取得する変数

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
項目がサポートされているかどうかを確認するには **DataSourceName** プロパティを設定する必要があります。
DataSourceName プロパティに空文字列を指定する場合は、予め **Select** メソッドを実行しておいてください。
(**OpenDataSource** メソッドを事前に実行した場合は **DataSourceName** プロパティは無効で、オープンされたデータソースが有効)

IsCapSupported メソッドを実行するには、事前に Initialize メソッドが実行されていなければなりません。
IsCapSupported メソッドを実行する前に OpenDataSource メソッドを実行すると、オープン中のデータソースは他のアプリケーションからは利用できなくなり、データソースを占有することができます。

Value に設定される値について
CapNo がドキュメントフィーダー機能を示す場合、0:フィーダー使用不可(原稿台使用可), 1:フィーダー使用可, 2: 原稿台とフィーダー使用可。
CapNo がドキュメントフィーダーに用紙があるかどうかの機能を示す場合、0:用紙なし, 1:用紙あり。
CapNo がインジケータを示す場合、0:インジケータ使用不可, 1:インジケータ使用可。
CapNo が UI コントロール機能を示す場合、0:メーカ提供 UI のみ, 1:メーカ提供 UI 非表示可。
CapNo がデバイスの稼動状態の確認機能を示す場合、現在の状態を表します。(0:応答なし, 1:使用可)
CapNo がサムネイル画像の転送機能を示す場合、0:サムネイル転送不可, 1:サムネイル転送可。
CapNo がドキュメントフィーダーの両面機能を示す場合、0:両面スキャン不可, 1:1 回の読み取りで両面スキャン、2:2 回の読み取りで両面スキャン。
CapNo が設定用 UI 表示機能を示す場合、0:設定用 UI 表示不可, 1:設定用 UI 表示可。
CapNo が UI 設定値の取得/設定機能を示す場合、0:UI 設定値の取得/設定不可, 1: UI 設定値の取得/設定可。
CapNo が明るさの自動化を示す場合、0:明るさの自動不可, 1:明るさの自動可。
CapNo が未定義サイズを示す場合、一般的には「0:未定義サイズ使用不可, 1:未定義サイズ使用可」となりますが、**エプソン製スキャナ用ドライバご利用時**には次の値が設定されます。「0:使用不可, 1:原稿台のみ可, 2:ADF のみ可, 3:原稿台,ADF とも可」。
CapNo がデジタルカメラ内の画像枚数を示す場合、撮影した画像の数を表します。
CapNo が自動領域切り出し機能を示す場合、0:領域切り出し不可, 1:領域切り出し可。
CapNo がデスキュー(傾き補正)を示す場合、0:デスキュー使用不可, 1:デスキュー使用可, 2:デスキュー滑らかオプション使用可。**2 が設定されるのはパナソニック製スキャナ用ドライバのみ**ですが、その場合は 1 も使用可能です。
CapNo がモアレフィルタ使用時の最大有効解像度を示す場合、最大有効解像度を表します。**エプソン製スキャナ用ドライバご利用時に有効です**。
CapNo が ADF を使用した両面スキャン時の裏面回転を示す場合、0:裏面回転不可, 1:裏面回転可。**キヤノン製 DR スキャナ用ドライバ、エプソン製スキャナ用ドライバ『EPSON Scan』ご利用時に有効です**。
CapNo が白黒/カラー自動判別読み取り機能を示す場合、0:白黒/カラー自動判別読み取り不可, 1:白黒/カラー自動判別読み取り可。**キヤノン製 DR スキャナ用ドライバ、エプソン製スキャナ用ドライバ、パナソニック製スキャナ用ドライバ、PFU 製スキャナ用ドライバご利用時に有効です**。
CapNo がマルチストリームを示す場合、0:マルチストリーム不可, 1:マルチストリーム可。マルチストリームとは 1 枚の原稿から 2 つの異なるピクセルタイプの画像を取得することです。**キヤノン DR 製スキャナ用ドライバ、パナソニック製スキャナ用ドライバ、PFU 製スキャナ用ドライバ、エプソン製スキャナ用ドライバご利用時に有効です**。
CapNo が文字向き検知または自動回転を示す場合、0:文字向き検知または自動回転不可, 1:文字向き検知または自動回転可。文字向き検知については**キヤノン製 DR スキャナ用ドライバ、パナソニック製スキャナ用ドライバご利用時に有効です**。
キヤノン製 DR スキャナ用ドライバでは「文字向き検知」、パナソニック製スキャナ用ドライバでは「原稿方向補正」という表現を用いています。
また、TWAIN 仕様書に記載されている自動回転に対応しているドライバについても確認できます。
CapNo がパンチ穴除去を示す場合、0:パンチ穴除去不可, 1:パンチ穴除去可。**キヤノン製 DR スキャナ用ドライバ、エプソン製スキャナ用ドライバ、パナソニック製スキャナ用ドライバ、PFU 製スキャナ用ドライバご利用時に有効です**。
CapNo が境界補正を示す場合、0:境界補正不可, 1:境界補正可。**エプソン製スキャナ用ドライバ、パナソニック製スキャナ用ドライバご利用時に有効です**。
CapNo がガンマ補正を示す場合、0:ガンマ補正不可, 1:ガンマ補正可。**エプソン製スキャナ用ドライバご利用時に有効です**。

※メソッドの戻り値が False の場合には Value は設定されません。

IsCapSupported メソッドは以前の ImageKit で提供していたスキャンコントロールの ScanIsCapSupported メソッドに相当します。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました。

【ImageKit7 ActiveX/VCL との違い】
CapNo に設定用 UI 表示機能、UI 設定値の取得/設定機能、マルチストリーム、文字向き検知または自動回転、パンチ穴除去の値が追加されました。また、ドキュメントフィーダー機能で Value に新たに 2 が設定されるようになりました。

【ImageKit8 ActiveX/VCL との違い】
CapNo に境界補正、ガンマ補正の値が追加されました。

【ImageKit9 ActiveX/VCL との違い】
CapNo に明るさの自動化の値が追加されました。

IsOpenDataSource（イメージキットコントロール／Scan メソッド）

【機能】
データソースがオープンされているかどうかを確認します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Scan**->**IsOpenDataSource**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Scan.IsOpenDataSource**

【引数】
ありません。

【戻り値】
オープンされている場合は True、そうでない場合は False を返します。

【解説】
事前に **OpenDataSource** メソッドが実行され、データソースが正しくオープンされている場合は True を返します。

IsOpenDataSource メソッドは以前の ImageKit で提供していたスキャンコントロールの **IsOpenDS** メソッドに相当します。

List(イメージキットコントロール／Scan メソッド)

【機能】
データソースを列挙します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Scan**->**List**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Scan.List**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
成功した場合は、データソースのリストが **DataSourceNameList** プロパティに設定されます。
DataSourceIndex プロパティは **DataSourceNameList** プロパティに設定されたデータソースの中で、現在選択されているデータソースが何番かを示します。最初のデータソースを示す場合は 0 となります。

List メソッドを実行するには、事前に **Initialize** メソッドが実行されていなければなりません。
List メソッドは以前の ImageKit で提供していたスキャンコントロールの **ScanList** メソッドに相当します。

カレントデータソースに位置付ける例:
(1)C++Builder
```
bool Ret;

Ret = VImageKit1->Scan->List();
if (Ret == false || VImageKit1->Scan->DataSourceNameList->Count == 0) return;
ComboBox1->Items = VImageKit1->Scan->DataSourceNameList;
ComboBox1->ItemIndex = VImageKit1->Scan->DataSourceIndex;
```
(2)Delphi
```
Ret: Boolean;

Ret := VImageKit1.Scan.List;
if (Ret = False) or (VImageKit1.Scan.DataSourceNameList.Count = 0) then Exit;
ComboBox1.Items := VImageKit1.Scan.DataSourceNameList;
ComboBox1.ItemIndex := VImageKit1.Scan.DataSourceIndex;
```

LoadTwain(イメージキットコントロール／Scan メソッド)

【機能】
TWAIN DLL をロードします。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Scan**->**LoadTwain**(const UnicodeString FileName)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Scan.LoadTwain**(const FileName: string)

【引数】

名称	内容
FileName	TWAIN DLL のファイル名 (フルパス)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
32 ビット用のパッケージで twain_32.dll(Windows フォルダ)を使用する場合や 64 ビット用のパッケージで 64 ビット用の TWAINDSM.dll(Windows¥System32 フォルダ)を使用する場合は当メソッドを実行する必要はありません(ImageKit 内で自動的にロード/解放が行われます)。
32 ビット用のパッケージで 32 ビット用の TWAINDSM.dll を使用する場合に当メソッドを実行してください。
LoadTwain メソッドは **Initialize** メソッドより前に実行してください。

Select（イメージキットコントロール／Scan メソッド）

【機能】
データソースの選択を実行します。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Scan**->**Select**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Scan.Select**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
当メソッドを実行すると、TWAIN で用意されているデータソースの選択用ダイアログが表示され、データソースの選択が可能になります。
成功した場合、**Manufacturer**,**ProductFamily**,**ProductName**,**ProtocolMajor**,**ProtocolMinor**,**SourceMajor**,**SourceMinor**,**SourceVersionInfo** プロパティに選択されたデータソースの情報が設定されます。

Select メソッドを実行するには、事前に **Initialize** メソッドが実行されていなければなりません。
Select メソッドは以前の ImageKit で提供していたスキャンコントロールの **ScanSelect** メソッドに相当します。

Terminate(イメージキットコントロール／Scan メソッド)

【機能】
TWAIN の終了処理を行います。

【書式】
(1)C++Builder　[*bool =*]*imagekitcontrolname*->**Scan**->**Terminate**()
(2)Delphi　[*Boolean =*]*imagekitcontrolname*.**Scan.Terminate**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
Initialize メソッドと対で使用してください。
Terminate メソッドは以前の ImageKit で提供していたスキャンコントロールの **ScanTerminate** メソッドに相当します。

Vector（イメージキットコントロール／カスタム階層プロパティ）

【機能】
ベクトルイメージの作成およびラスタとベクトルの相互変換機能を提供します。

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
ButtonName	RasterToVector メソッド処理中のダイアログボックスに表示するボタンの名称を設定
Cancel	RasterToVector メソッド処理の中止の設定
Caption	RasterToVector メソッド処理中のダイアログボックスに表示するタイトルバーの名称を設定
Message	RasterToVector メソッド処理中のダイアログボックスに表示するメッセージを設定

●メソッド一覧（アルファベット順）

カスタムメソッド	内容
CreateImage	ベクトルイメージを新規に作成
RasterToVector	ラスタイメージをベクトルイメージに変換
VectorToRaster	ベクトルイメージをラスタイメージに変換

ButtonName,Caption,Message(イメージキットコントロール／Vector プロパティ)

【機能】
RasterToVector メソッド処理中のダイアログボックスに表示するボタン、キャプション、メッセージを取得または設定します。

【書式】
※ButtonName にて説明(その他も同様な使い方)
(1)C++Builder　*imagekitcontrolname*->Vector->ButtonName [= *UnicodeString*]
(2)Delphi　*imagekitcontrolname*.Vector.ButtonName [= *string*]

【設定値】
ButtonName プロパティは、処理中のダイアログボックスのボタンに表示する文字列。
Caption プロパティは、処理中のダイアログボックスのタイトルバーに表示する文字列。
Message プロパティは、処理中のダイアログボックスの中央に表示する文字列。

【解説】
ButtonName,Caption,Message プロパティ全てに何も設定しなかった場合は、進捗状況のダイアログボックスは表示されず、Progress イベントが発生します。
逆にどれか一つでも有効な文字列を設定した場合は、進捗状況ダイアログボックスが表示され、Progress イベントは発生しません。

【値の設定】 実行時

【値の参照】 実行時

Cancel(イメージキットコントロール／Vector プロパティ)

【機能】
RasterToVector メソッド処理を中止するかどうかを設定します。

【書式】
(1)C++Builder　*imagekitcontrolname*->Vector->Cancel [= *bool*]
(2)Delphi　*imagekitcontrolname*.Vector.Cancel [= *Boolean*]

【設定値】

値	説明
True	処理を中止する
False	処理を中止しない

【解説】
Cancel プロパティに True を設定すると、処理を中止できます。

【値の設定】 実行時

【値の参照】 不可

CreateImage(イメージキットコントロール／Vector メソッド)

【機能】
新規にベクトルイメージを作成します。

【書式】
(1)C++Builder
[*NativeUInt =*]*imagekitcontrolname*->**Vector**->**CreateImage**(TVIkCreateVectImage AType, int AWidth, int AHeight, int Xdpi, int Ydpi)
(2)Delphi
[*THandle =*]*imagekitcontrolname*.**Vector.CreateImage**(AType: TVIkCreateVectImage; AWidth, AHeight: Integer; Xdpi, Ydpi: Integer)

【TVIkCreateVectImage 型】
ユニット
IkInit
type
TVIkCreateVectImage = (vikCreateWMF, vikCreateEMF, vikCreateDXF, vIkCreateSVG, vikCreateSVG);

【引数】

名称	内容
AType	作成するベクトルイメージのタイプ(vikCreateWMF:WMF, vikCreateEMF:EMF, vikCreateDXF:DXF, vIkCreateSVG:SVG, vikCreateSVG:SXF)
AWidth	作成するイメージの幅(ピクセル)
AHeight	作成するイメージの高さ(ピクセル)
Xdpi	作成するイメージの横方向の 1 インチあたりのピクセル数
Ydpi	作成するイメージの縦方向の 1 インチあたりのピクセル数

【戻り値】
成功の場合は作成したベクトルイメージのメモリハンドル、失敗の場合は 0 を返します。

【解説】
CreateImage メソッドは以前の ImageKit で提供していたコモンコントロールの **CreateVectImageEx** メソッドに相当します。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikCreateWMF, ikCreateEMF, ikCreateDXF, ikCreateSVG, ikCreateSXF)。

RasterToVector(イメージキットコントロール／Vector メソッド)

【機能】
ラスタイメージをベクトルイメージに変換します。

【書式】
(1)C++Builder
[*bool* =]*imagekitcontrolname*->**Vector**->**RasterToVector**(bool BlackOnWhite, short Tolerance, int TimeOutSeconds)
(2)Delphi
[*Boolean* =]*imagekitcontrolname*.**Vector**.**RasterToVector**(BlackOnWhite: Boolean; Tolerance: Smallint; TimeOutSeconds: Integer)

【引数】

名称	内容
BlackOnWhite	False:黒地に白字、True:白地に黒字
Tolerance	許容値 0(感度高)～10(感度低)
TimeOutSeconds	処理を中止するタイムアウト(秒)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
白黒 2 値イメージが対象です。
処理対象となるラスタイメージのメモリハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(**LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
成功した場合、ラスタイメージのメモリハンドルは解放され、変換されたベクトルイメージのメモリハンドルが **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。
処理中に TimeOutSeconds で設定された時間を超えた場合、強制的に処理を中断します。

Caption,**Message**,**ButtonName** プロパティが空の文字列の場合は、処理中のダイアログボックスは表示されません。ダイアログボックスが表示される場合は、処理進捗状況がゲージ上に%形式で表示されます。

RasterToVector メソッドは以前の ImageKit で提供していたエフェクトコントロールの **RasterToVector** メソッドに相当します。

VectorToRaster(イメージキットコントロール／Vector メソッド)

【機能】
ベクトルイメージをラスタイメージに変換します。

【書式】
(1)C++Builder
[*bool =*]*imagekitcontrolname*->**Vector**->**VectorToRaster**(short BitCount, bool DxfBlack, Byte BackRed, Byte BackGreen, Byte BackBlue)
(2)Delphi
[*Boolean =*]*imagekitcontrolname*.**Vector.VectorToRaster**(BitCount: Smallint; DxfBlack: Boolean; BackRed: Byte; BackGreen: Byte; BackBlue: Byte)

【引数】

名称	内容
BitCount	ラスタイメージに変換する際のビット数(1,4,8,16,24,32)
DxfBlack	イメージが DXF を表す場合、白を黒に置き換えて描画 [True:する False:しない]
BackRed	背景色(赤)
BackGreen	背景色(緑)
BackBlue	背景色(青)

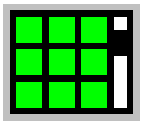
【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
処理対象となるベクトルイメージのメモリハンドルを **ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティに設定します(**LayerNo** プロパティによって処理されるイメージハンドルが決まります)。
成功した場合、ベクトルイメージのメモリハンドルは解放され、変換されたラスタイメージのメモリハンドルが **LayerNo** プロパティの示すプロパティ(**ImageHandle** プロパティもしくは **Layer[LayerNo].ImageHandle** プロパティ)に設定されます。

VectorToRaster メソッドは以前の ImageKit で提供していたコモンコントロールの **VectToDib** メソッドに相当します。

【ImageKit7/8/9/10 ActiveX との違い】
引数の BackRed,BackGreen,BackBlue が Byte 型に変更されました。

1－2. サムネイルコントロール



サムネイルコントロール
フォルダとファイルタイプなどを指定して、複数のイメージファイルを自動的にサムネイル表示します。
マウスクリックやマウスの移動、およびキーボードやメソッドからサムネイル画像の選択ができます。

継承
TCustomControl
ユニット
IkThumb

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Appearance	サムネイルコントロールの縁の設定
BorderColor	サムネイルコントロールの縁の色
BorderVisible	サムネイルコントロールの縁の描画設定
Button3D	サムネイルコントロールのボタンの 3D 設定
Cancel	処理の中止の設定
Canvas.Handle	サムネイルコントロールのデバイスコンテキスト
ClearOnLoad	サムネイルコントロールに画像をロードする時のモード
ColCount	表示行数の設定
Color	サムネイルコントロールの背景色
Cursor	サムネイルコントロールの描画領域内におけるマウスカーソルの形状
DispFileName	ファイル名の表示設定
EdgeSize	サムネイルコントロールの縁のサイズ
EnableArrowKeys	矢印キーおよび PageUp,PageDown キーを有効にするかどうかの設定
Enabled	マウスやキーボードイベントへの応答可否
EnableDragSource	サムネイル画像のドラッグ&ドロップの設定
EnableDragTarget	サムネイルコントロールへのドラッグ&ドロップの設定
EnableMouseMoveButton	マウスの移動に伴うボタンの設定
EnableMouseWheel	マウスホイールの設定
EnableMove	サムネイル画像の移動可否
EnableSelectMultiFiles	複数のサムネイル画像を選択するかどうかの設定
ErrorStatus	エラーの種類を示す
ExifAutoRotate	読み込み対象ファイルが Exif 形式の場合、主画像の方向に合わせて回転を行って表示する
FileExt	読み込むファイルタイプを設定
FileName	選択したサムネイル画像のファイル名
FilePath	読み込むフォルダを設定
ForeColor	サムネイルのボタンの色を設定
ForeColorDown	サムネイルのボタンの色を設定
GapSize	サムネイル画像とその間の間隔を設定
Handle	サムネイルコントロールのウィンドウハンドル
ImageHandle	サムネイルコントロールにロードするメモリハンドル
ImageSize	サムネイル画像のサイズを設定
ListOfFileNames	サムネイルコントロールで表示するイメージファイルの名称リストを取得または設定
Picture	サムネイルのボタンの背景画像
PictureDown	サムネイルのボタンの背景画像
PictureDownFile	サムネイルのボタンの背景画像を示すファイル名
PictureFile	サムネイルのボタンの背景画像を示すファイル名
ReadMode	読み込みのモードを設定
RowCount	表示列数の設定
ScrollBar	スクロールバーの表示設定
ScrollStep	スクロールの移動量を設定
ScrollVH	スクロールの方向の設定
SelectCursor	サムネイルのコマを選択する際のマウスカーソルの形状

SelectTextBackColor	選択されたサムネイル画像のファイル名および注釈の背景色
SelectTextColor	選択されたサムネイル画像のファイル名および注釈を描画色
SelectTextTransparent	選択されたサムネイル画像のファイル名および注釈を描画する時の透過設定
Sort	フォルダのファイルのソート方法を設定
SortDate	Sort プロパティに日付が含まれる時のソートする日付
SortOrder	フォルダのファイルのソート順を設定
StartNum	表示を開始するイメージの番号を取得もしくは設定
Style	サムネイルの背景のデザインを設定
Text	サムネイル画像の注釈の設定
TextBackColor	サムネイル画像のファイル名および注釈の背景色
TextColor	サムネイル画像のファイル名および注釈を描画色
TextTransparent	サムネイル画像のファイル名および注釈を描画する時の透過設定
ThumbArrayFileName	指定したサムネイル画像のファイル名
ThumbArrayNum	サムネイル画像のイメージ番号を設定
ThumbArrayPage	指定したサムネイル画像のファイルのページ番号
ThumbArrayPathName	指定したサムネイル画像のフォルダ名
ThumbnailFile	サムネイルファイルの設定
Touch	指によるスクロールの適用可否

【ImageKit5 ActiveX との違い】
変更されたプロパティ:
BackColor --> Color
Hwnd --> Handle

【ImageKit6 ActiveX との違い】
削除されたプロパティ:EnableOLEDrag, MouseCursorTypeFile
※MouseCursorTypeFile プロパティは Cursor プロパティで代用

変更されたプロパティ:
BackColor --> Color
ForeColorUp --> ForeColor
Hdc --> Canvas.Handle
Hwnd --> Handle
MouseCursor --> SelectCursor
MouseCursorType --> Cursor
PictureUp --> Picture
PictureUpFile --> PictureFile

【ImageKit7/8/9/10 ActiveX との違い】
削除されたプロパティ:EnableOLEDrag, MouseCursorFile, SelectMouseCursorFile
※MouseCursorTypeFile プロパティは Cursor プロパティで代用
※SelectMouseCursorTypeFile プロパティは SelectCursor プロパティで代用

変更されたプロパティ:
BackColor --> Color
ForeColorUp --> ForeColor
Hdc --> Canvas.Handle
Hwnd(ImageKit7/8 ActiveX) --> Handle
HWND(ImageKit9 ActiveX) --> Handle
MouseCursorType --> Cursor
PictureUp --> Picture
PictureUpFile --> PictureFile
SelectMouseCursorType --> SelectCursor

【ImageKit9/10 ActiveX との違い】
変更されたプロパティ:
EnableTouch --> Touch

【ImageKit6 VCL との違い】

変更されたプロパティ:
ForeColorUp --> ForeColor
PictureUp --> Picture
PictureUpFile --> PictureFile
ThumbSelectCursor --> SelectCursor

【ImageKit7/8 VCL との違い】
変更されたプロパティ:
ForeColorUp --> ForeColor
PictureUp --> Picture
PictureUpFile --> PictureFile

●メソッド一覧(アルファベット順)

カスタムメソッド	内容
Clear	サムネイル画像の表示をクリア
Delete	指定したサムネイル画像の解除
DeSelectImage	指定したサムネイル画像の選択を解除
Display	サムネイルイメージを表示
GetFiles	サムネイルイメージの取得
IsSelected	指定したサムネイル画像の選択可否
Refresh	コントロール上のイメージの再描画
Scroll	スクロールを実行
SelectImage	指定したサムネイル画像の選択
SortImage	サムネイルイメージをソートします。
UpdateThumbnailFile	サムネイルファイルを更新

●イベント一覧(アルファベット順)

カスタムイベント	内容
DropFileInThumb	サムネイルコントロールにファイルをドロップした時に発生
EndLoadFile	サムネイルコントロール上の 1 イメージをロードした後に発生
MouseMoveOnThumb	サムネイルコントロール上をマウスで移動した時に発生
SelectFile	サムネイルコントロール上のイメージを選択すると発生
ShowThumbImage	サムネイル画像を表示した後に発生
StartLoadFile	サムネイルコントロール上の 1 イメージをロードする前に発生

Appearance(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールの縁の表示を設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Appearance** [= *TVIkAppearance*]
(2)Delphi　*thumbcontrolname*.**Appearance** [= *TVIkAppearance*]

【TVIkAppearance 型】
ユニット
IkInit
type
TVIkAppearance = (vikFlat, vikLowered, vikRaised);

【設定値】

値	説明
vikFlat	Flat
vikLowered	凹
vikRaised	凸

【解説】
BorderVisible プロパティが True の場合に有効です。
vikFlat の場合、**BorderColor** プロパティに設定されている色で縁を描画します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit6/7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikFlat, ikLowered, ikRaised)。

BorderColor,Color(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールの色情報を指定します。

【書式】
※**BorderColor** にて説明(**Color** も同様な使い方)
(1)C++Builder　　*thumbcontrolname*->**BorderColor** [= *TColor*]
(2)Delphi　　*thumbcontrolname*.**BorderColor** [= *TColor*]

【設定値】
BorderColor はサムネイルコントロールの縁の色。
Color はサムネイルコントロールの背景色。

【解説】
値を設定する場合は、色定数(clRed など)や **RGB**(Red,Green,Blue)の戻り値などを与えます。
BorderColor プロパティは **Appearance** プロパティが vikFlat で **BorderVisible** プロパティが True の場合に有効です。
TColor 型については Delphi や C++Builder のヘルプを参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit5/6/7/8/9/10 ActiveX との違い】
Color プロパティの名称が **BackColor** から変更されました。

BorderVisible(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールの縁を描画するかどうかを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**BorderVisible** [= *bool*]
(2)Delphi　*thumbcontrolname*.**BorderVisible** [= *Boolean*]

【設定値】

値	説明
True	縁を描画する
False	縁を描画しない

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Button3D(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールのコマ(サムネイル画像を表示する領域)を 3D にするかどうかを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Button3D** [= *bool*]
(2)Delphi　*thumbcontrolname*.**Button3D** [= *Boolean*]

【設定値】

値	説明
True	3D にする
False	3D にしない

【解説】
Style プロパティが vikButton(ボタン)の時に有効です。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Cancel(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールの表示処理を中止するかどうかを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Cancel** [= *bool*]
(2)Delphi　*thumbcontrolname*.**Cancel** [= *Boolean*]

【設定値】

値	説明
True	処理を中止する
False	処理を中止しない

【解説】
Cancel プロパティに True を設定すると、表示処理を中止できます。

【値の設定】 実行時

【値の参照】 不可

Canvas.Handle(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールのデバイスコンテキストを示します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Canvas**->**Handle** [= *HDC*]
(2)Delphi　*thumbcontrolname*.**Canvas**.**Handle** [= *HDC*]

【参照値】
サムネイルコントロールのデバイスコンテキスト。

【解説】
Canvas については Delphi や C++Builder のヘルプの TCanvas を参照してください。

【値の設定】 不可

【値の参照】 実行時

【ImageKit6/7/8/9/10 ActiveX との違い】
プロパティの名称が **Hdc** から変更されました。

ClearOnLoad(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールに画像をロードする時のモードを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ClearOnLoad** [= *bool*]
(2)Delphi　*thumbcontrolname*.**ClearOnLoad** [= *Boolean*]

【設定値】

値	説明
True	ロードする時に前回の結果をクリアする(デフォルト)
False	ロードする時に前回の結果を保持する

【解説】
False を設定すると、前回ファイルなどをロードした結果に対して画像を追加することができます。例えば、ADF を使用して原稿を連続してスキャンする場合やデータベースから 1 つずつ画像を読み込む場合などが考えられます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ColCount,RowCount（サムネイルコントロール／カスタムプロパティ）

【機能】
サムネイルコントロールで表示する行と列を設定します。

【書式】
※ColCount にて説明（RowCount も同様な使い方）
(1)C++Builder　　*thumbcontrolname*->ColCount [= *short*]
(2)Delphi　　*thumbcontrolname*.ColCount [= *Smallint*]

【設定値】
ColCount は表示列数（1～）
RowCount は表示行数（1～）

【解説】
ColCount、RowCount プロパティで設定された値の積が 1 画面に表示されるイメージの数となります。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Cursor,SelectCursor(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールのマウスカーソルの形状を設定します。

【書式】
※**Cursor** にて説明(**SelectCursor** も同様な使い方)
(1)C++Builder　　*thumbcontrolname*->**Cursor** [= *TCursor*]
(2)Delphi　　*thumbcontrolname*.**Cursor** [= *TCursor*]

【設定値】
crDefault や crArrow など。詳しくは Delphi や C++Builder のヘルプを参照のこと。

【解説】
Cursor プロパティはサムネイルコントロールの描画領域内におけるマウスカーソルの形状を表します。
SelectCursor プロパティはサムネイルのコマを選択する際のマウスカーソルの形状を表します。
Cursor プロパティのデフォルト値は crDefault で、**SelectCursor** プロパティのデフォルト値は crArrow です。
カスタムカーソルを割り当てる場合は、Delphi や C++Builder のヘルプの TControl.Cursor の例を参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit6 ActiveX との違い】
MouseCursorType, **MouseCursorTypeFile** プロパティの機能が **Cursor** プロパティに統一されました。
MouseCursor プロパティの名称が **SelectCursor** プロパティに変更され、型も文字列型から TCursor 型に変更されました。

【ImageKit7/8/9/10 ActiveX との違い】
MouseCursorType, **MouseCursorFile** プロパティの機能が **Cursor** プロパティに統一されました。
SelectMouseCursorType, **SelectMouseCursorFile** プロパティの機能が **SelectCursor** プロパティに統一されました。

【ImageKit6 VCL との違い】
ThumbSelectCursor プロパティの名称が **SelectCursor** に変更されました。

DispFileName(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールに表示する画像の下にそのファイル名を表示するかどうかを指定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**DispFileName** [= *bool*]
(2)Delphi　*thumbcontrolname*.**DispFileName** [= *Boolean*]

【設定値】

値	説明
True	ファイル名もしくは注釈を表示する
False	ファイル名もしくは注釈を表示しない

【解説】
True を設定すると、サムネイル画像の下にファイル名や注釈を表示することができます。
ファイル名を表示する場合は当プロパティを True にするだけで構いませんが、注釈を表示する場合は **ShowThumbImage** イベント内で **Text** プロパティに注釈を設定してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

EdgeSize(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールの縁のサイズを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**EdgeSize** [= *short*]
(2)Delphi　*thumbcontrolname*.**EdgeSize** [= *Smallint*]

【設定値】
縁からコマ(サムネイル画像を表示する領域)の始まる位置までのサイズ(ピクセル)

【値の設定】 デザイン時、実行時

【値の参照】 実行時

EnableArrowKeys(サムネイルコントロール／カスタムプロパティ)

【機能】
矢印キーおよび PageUp,PageDown キーを有効にするかどうかを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**EnableArrowKeys** [= *bool*]
(2)Delphi　*thumbcontrolname*.**EnableArrowKeys** [= *Boolean*]

【設定値】

値	説明
True	矢印キーと PageUp,PageDown キーを有効
False	矢印キーと PageUp,PageDown キーを無効

【解説】
True を設定すると、矢印キーや PageUp,PageDown キーでサムネイル画像を操作できます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Enabled(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールのマウスやキーボードイベントに応答するかどうかを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Enabled** [= *bool*]
(2)Delphi　*thumbcontrolname*.**Enabled** [= *Boolean*]

【設定値】

値	説明
True	マウスやキーボードイベントを有効
False	マウスやキーボードイベントを無効

【解説】
True を設定するとマウスダウンやキーダウンなどのイベントに応答できますが、False の場合は該当イベントが発生しません。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

EnableDragSource(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイル画像をイメージキットコントロールや他のサムネイルコントロールにドラッグ&ドロップできるかどうかを指定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**EnableDragSource** [= *bool*]
(2)Delphi　*thumbcontrolname*.**EnableDragSource** [= *Boolean*]

【設定値】

値	説明
True	サムネイル画像をイメージキットコントロールや他のサムネイルコントロールにドラッグ&ドロップ可
False	サムネイル画像をイメージキットコントロールや他のサムネイルコントロールにドラッグ&ドロップ不可

【解説】
True を設定すると、サムネイル画像をイメージキットコントロールや他のサムネイルコントロールにドラッグ&ドロップできます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

EnableDragTarget(サムネイルコントロール／カスタムプロパティ)

【機能】
Windows のエクスプローラなどから画像をドロップできるかどうかを指定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**EnableDragTarget** [= *bool*]
(2)Delphi　*thumbcontrolname*.**EnableDragTarget** [= *Boolean*]

【設定値】

値	説明
True	画像をドロップ可
False	画像をドロップ不可

【解説】
True を設定すると、サムネイルコントロールへ画像をドロップできます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

EnableMouseMoveButton(サムネイルコントロール／カスタムプロパティ)

【機能】
マウスの移動に伴いコマ(サムネイル画像を表示する領域)を自動的に押下するかどうかを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**EnableMouseMoveButton** [= *bool*]
(2)Delphi　*thumbcontrolname*.**EnableMouseMoveButton** [= *Boolean*]

【設定値】

値	説明
True	コマを自動的に押下(選択状態にする)
False	コマを手動で押下

【解説】
True を設定しても、手動でコマを押すことは可能です。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

EnableMouseWheel(サムネイルコントロール／カスタムプロパティ)

【機能】
マウスホイールを有効にするかどうかを指定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**EnableMouseWheel** [= *bool*]
(2)Delphi　*thumbcontrolname*.**EnableMouseWheel** [= *Boolean*]

【設定値】

値	説明
True	マウスホイールを有効
False	マウスホイールを無効

【値の設定】 デザイン時、実行時

【値の参照】 実行時

EnableMove(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールのサムネイル画像の移動可否を設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**EnableMove** [= *bool*]
(2)Delphi　*thumbcontrolname*.**EnableMove** [= *Boolean*]

【設定値】

値	説明
True	サムネイル画像の移動可
False	サムネイル画像の移動不可

【解説】
デフォルトは False です。True の場合、マウス操作でサムネイル画像の移動が可能です。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

EnableSelectMultiFiles（サムネイルコントロール／カスタムプロパティ）

【機能】
複数のサムネイル画像を選択するかどうかを指定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**EnableSelectMultiFiles** [= *bool*]
(2)Delphi　*thumbcontrolname*.**EnableSelectMultiFiles** [= *Boolean*]

【設定値】

値	説明
True	複数のサムネイル画像を選択可
False	単一のサムネイル画像のみ選択可

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ErrorStatus(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールを使用して起きたエラーの種類を示します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ErrorStatus** [= *short*]
(2)Delphi　*thumbcontrolname*.**ErrorStatus** [= *Smallint*]

【参照値】

値	説明
0	正常終了(エラーなし)
1	キャンセル
2	その他のエラー
3	メモリエラー
4	サポート外機能
5	設定値が不正
6	ファイルが不正
7	イメージが不正
9	描画エラー
10	対象外イメージ
11	実行順序エラー
14	DLL がロードされていない
15	通信エラー
17	ファイルが壊れている
18	タイムオーバーで処理をキャンセル
19	パスワード、ユーザ名が異なる(アクセスができない)
20	ファイルがオープンできない
21	ファイルが作成できない
22	ファイルが書き込みできない
23	ファイルが読み込みできない
24	表示イメージデータがない

【解説】
サムネイルコントロールのメソッドを実行することにより、値を参照できます。

【値の設定】 不可

【値の参照】 実行時

ExifAutoRotate(サムネイルコントロール／カスタムプロパティ)

【機能】
読み込み対象ファイルが Exif 形式の場合、主画像の方向に合わせて回転を行って表示します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ExifAutoRotate** [= *bool*]
(2)Delphi　*thumbcontrolname*.**ExifAutoRotate** [= *Boolean*]

【設定値】

値	説明
True	主画像の方向に合わせて回転して表示する
False	主画像をありのまま表示する(方向を無視して表示)

【解説】
デフォルトは True です。当プロパティは読み込み対象ファイルが Exif 形式以外の場合は無効です。

【値の設定】 実行時

【値の参照】 不可

FileExt,FilePath（サムネイルコントロール／カスタムプロパティ）

【機能】
サムネイルコントロールで表示するイメージのフォルダやファイルのタイプを設定します。

【書式】
※**FileExt** にて説明（**FilePath** も同様な使い方）
(1)C++Builder　　*thumbcontrolname*->**FileExt** [= *UnicodeString*]
(2)Delphi　　*thumbcontrolname*.**FileExt** [= *string*]

【設定値】
FileExt はファイルのタイプ。（ファイル名も可）
FilePath はフォルダ名。

【解説】
FileExt に複数のタイプを指定する場合や **FilePath** に複数のフォルダを指定する場合は、ファイルの拡張子やフォルダの名称をセミコロン（;）で区切ってください。
また、**FileExt** はアルファベットの大文字と小文字を区別しません。（"BMP"と"bmp"を同じとみなします）
例
```
VIkThumb1.FileExt := 'BMP;JPG;PNG;';
```

FileExt にファイル名を指定する場合は、"*1.bmp"や"001.bmp"のように設定してください。ただし、"*.bmp;001.bmp"のようにワイルドカードを使用した場合は同じ拡張子を複数設定できませんので注意してください。
例
```
正：VIkThumb1.FileExt := '002.bmp;001.bmp';
誤：VIkThumb1.FileExt := '*.bmp;001.bmp';
```

フォルダ内の特定のファイルのみ表示する場合は **ListOfFileNames** プロパティに表示するファイル名を設定し、**FilePath** プロパティに空文字列を設定してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

FileName(サムネイルコントロール／カスタムプロパティ)

【機能】
該当するサムネイル画像を選択すると、そのファイル名がセットされます。

【書式】
(1)C++Builder　*thumbcontrolname*->**FileName** [= *UnicodeString*]
(2)Delphi　*thumbcontrolname*.**FileName** [= *string*]

【参照値】
ファイル名。

【解説】
SelectFile イベントや **MouseMoveOnThumb** イベント(**EnableMouseMoveButton** プロパティが True の場合)でもファイル名を取得できます。
複数のサムネイル画像が選択された場合は、最後に選択されたファイルとなります。
サムネイル画像は、マウスクリックやマウスの移動、およびキーボードや **SelectImage** メソッドによって選択できます。

【値の設定】 不可

【値の参照】 実行時

ForeColor,ForeColorDown（サムネイルコントロール／カスタムプロパティ）

【機能】
サムネイルのコマ（サムネイル画像を表示する領域）の色を設定します。

【書式】
※**ForeColor** にて説明（**ForeColorDown** も同様な使い方）
(1)C++Builder　　*thumbcontrolname*->**ForeColor** [= *TColor*]
(2)Delphi　　*thumbcontrolname*.**ForeColor** [= *TColor*]

【設定値】
ForeColor はデフォルトの色。（コマが押されていない状態）
ForeColorDown はコマが押された時の色。

【解説】
値を設定する場合は、色定数(clRed など)や **RGB**(Red,Green,Blue)の戻り値を与えます。
Style プロパティが vikButton（ボタン）、vikFilm（フィルム）の時に有効です。
TColor 型については Delphi や C++Builder のヘルプを参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit6/7/8 ActiveX/VCL との違い】
ForeColorUp プロパティが **ForeColor** プロパティに変更されました。

GapSize(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルのコマの間の間隔を設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**GapSize** [= *short*]
(2)Delphi　*thumbcontrolname*.**GapSize** [= *Smallint*]

【設定値】
コマ間の隙間(ピクセル)

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Handle(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールのウィンドウハンドルを示します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Handle** [= *HWND*]
(2)Delphi　*thumbcontrolname*.**Handle** [= *HWND*]

【参照値】
サムネイルコントロールのウィンドウハンドル。

【値の設定】 不可

【値の参照】 実行時

【ImageKit5/6/7/8 ActiveX との違い】
プロパティの名称が **Hwnd** から変更されました。

【ImageKit9/10 ActiveX との違い】
プロパティの名称が **HWND** から変更されました。

ImageHandle(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールにロードするメモリハンドルを示します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ImageHandle** [= *NativeUInt*]
(2)Delphi　*thumbcontrolname*.**ImageHandle** [= *THandle*]

【設定値】
ラスタイメージもしくはベクトルイメージのメモリハンドル。

【解説】
ADF を使用して原稿を連続してスキャンする場合などに使用します。

例(Delphi):

```
procedure TForm1.VImageKit1AfterScan(Sender: TObject; DibHandle, OrgHandle: Cardinal; ImageCount: Integer;
BitOrder: Smallint);
begin
  VIkThumb1.ImageHandle := VImageKit1.CopyImage(DibHandle);
end;
```

【値の設定】 実行時

【値の参照】 不可

ImageSize(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイル画像をコントロールに描画する際のサイズを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ImageSize** [= *short*]
(2)Delphi　*thumbcontrolname*.**ImageSize** [= *Smallint*]

【設定値】
画像サイズ(ピクセル)
初期値は 96

【解説】
オリジナル画像の縦と横のサイズの大きい方のサイズに合わせます。ただし、サムネイル画像の描画領域が設定したサイズよりも小さい場合は、それに合わせます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ListOfFileNames(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールで表示するイメージファイルの名称リストを取得または設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ListOfFileNames** [= *TStringDynArray*]
(2)Delphi　*thumbcontrolname*.**ListOfFileNames** [= *TStringDynArray*]

【設定値】
ファイル名のリスト(一つでも可)。

【解説】
複数のフォルダに存在する特定のファイルや一つのフォルダでも特定のファイルのみ表示する場合に使用します。
ファイル名のリストを使用する場合は **FilePath** プロパティに空文字列を設定して無効にしてください。当プロパティが有効となる場合は **FileExt** プロパティの値は無視されます。また、当プロパティにはフルパスまたは相対パスでファイル名を設定してください。

【値の設定】 実行時

【値の参照】 実行時

Picture,PictureDown(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルのコマ(サムネイル画像を表示する領域)の背景画像を設定します。

【書式】
※**Picture** にて説明(**PictureDown** も同様な使い方)
(1)C++Builder　　*thumbcontrolname*->**Picture** [= *TPicture*]
(2)Delphi　　*thumbcontrolname*.**Picture** [= *TPicture*]

【設定値】
Picture はデフォルトの背景画像。(コマが押されていない状態)
PictureDown はコマが押された時の背景画像。

【解説】
Style プロパティが vikCustom(カスタム)の場合に有効です。
イメージの黒(Red=0,Green=0,Blue=0)の部分が透過となり、この部分にサムネイル画像が表示されます。
PictureDownFile,**PictureFile** プロパティもサムネイルのコマの背景画像を示すため、どちらか一方をご使用ください。

コードで設定する例:
(1)C++Builder
```
VIkThumb1->Picture->LoadFromFile("HalfOpen.bmp");
VIkThumb1->PictureDown->LoadFromFile("FullOpen.bmp");
```
(2)Delphi
```
VIkThumb1.Picture.LoadFromFile('HalfOpen.bmp');
VIkThumb1.PictureDown.LoadFromFile('FullOpen.bmp');
```

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit6/7/8 ActiveX/VCL との違い】
PictureUp プロパティが **Picture** プロパティに変更されました。

PictureDownFile,PictureFile（サムネイルコントロール／カスタムプロパティ）

【機能】
サムネイルのコマ（サムネイル画像を表示する領域）の背景画像を示すファイル名を設定します。

【書式】
※**PictureDownFile** にて説明（**PictureFile** も同様な使い方）
(1)C++Builder　*thumbcontrolname*->**PictureDownFile** [= *UnicodeString*]
(2)Delphi　*thumbcontrolname*.**PictureDownFile** [= *string*]

【設定値】
PictureDownFile はコマが押された時の背景画像を示すファイル名。
PictureFile はデフォルトの背景画像を示すファイル名。（コマが押されていない状態）

【解説】
Style プロパティが vikCustom（カスタム）の場合に有効です。
イメージの黒（Red=0,Green=0,Blue=0）の部分が透過となり、この部分にサムネイル画像が表示されます。
Picture,PictureDown プロパティもサムネイルのコマの背景画像を示すため、どちらか一方をご使用ください。

【値の設定】 実行時

【値の参照】 不可

【ImageKit6/7/8 ActiveX/VCL との違い】
PictureUpFile プロパティが **PictureFile** プロパティに変更されました。

ReadMode(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルとして表示するイメージの読み込みモードを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ReadMode** [= *TVIkThumbnailReadMode*]
(2)Delphi　*thumbcontrolname*.**ReadMode** [= *TVIkThumbnailReadMode*]

【TVIkThumbnailReadMode 型】
ユニット
IkThumb
type
TVIkThumbnailReadMode = (vikThumbRd, vikThumbRdDsp, vikThumbRdMultiImg, vikThumbRdDspMultiImg, vikThumbRdImage);

【設定値】

値	説明
vikThumbRd	読込
vikThumbRdDsp	読込と表示
vikThumbRdMultiImg	読込(マルチイメージ対応)
vikThumbRdDspMultiImg	読込と表示(マルチイメージ対応)
vikThumbRdImage	イメージハンドルの読込

【解説】
ReadMode プロパティを vikThumbRd, vikThumbRdMultiImg に設定した場合、**GetFiles** メソッドで予め **FilePath** プロパティに設定されたフォルダの該当するファイルまたは **ListOfFileNames** プロパティに設定されたファイルをメモリ上に読み込み、**Display** メソッドでコントロール上に表示します。
ReadMode プロパティを vikThumbRdDsp, vikThumbRdDspMultiImg に設定した場合、**Display** メソッドを実行すると **FilePath** プロパティに設定されたフォルダの該当するファイルまたは **ListOfFileNames** プロパティに設定されたファイルを読みながら、コントロール上に表示します。
※マルチイメージは GIF と TIFF ファイルが対象となります。

ReadMode プロパティを vikThumbRdImage に設定した場合、**Display** メソッドを実行すると **ImageHandle** プロパティで設定されたイメージをコントロール上に表示します。複数のイメージを同時に表示する場合は、**ClearOnLoad** プロパティを False に設定して **ImageHandle** プロパティにそれぞれのメモリハンドルを設定してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit6/7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikThumbRd, ikThumbRdDsp, ikThumbRdMultiImg, ikThumbRdDspMultiImg, ikThumbRdImage)。

ScrollBar(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールに対してスクロールバーの表示の有無を指定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ScrollBar** [= *bool*]
(2)Delphi　*thumbcontrolname*.**ScrollBar** [= *Boolean*]

【設定値】

値	説明
True	サムネイルコントロールにスクロールバーを表示する
False	サムネイルコントロールにスクロールバーを表示しない

【解説】
この **ScrollBar** プロパティを True にすることにより、サムネイルコントロール上に収まりきらないイメージを表示する際にスクロールバーを表示することができます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ScrollStep(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルのスクロール移動量を設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ScrollStep** [= *short*]
(2)Delphi　*thumbcontrolname*.**ScrollStep** [= *Smallint*]

【設定値】
スクロールする移動量(行または列単位)

【解説】
プロパティに 1 を設定すると先へ 1 つ進み、-1 にすると前に 1 つ戻ります。
スクロールする方向は、**ScrollVH** プロパティに依存します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ScrollVH(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルのスクロール方向を設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ScrollVH** [= *TVIkThumbnailScroll*]
(2)Delphi　*thumbcontrolname*.**ScrollVH** [= *TVIkThumbnailScroll*]

【TVIkThumbnailScroll 型】
ユニット
IkThumb
type
TVIkThumbnailScroll = (vikThumbVertical, vikThumbHorizontal);

【設定値】

値	説明
vikThumbVertical	垂直方向(縦)
vikThumbHorizontal	水平方向(横)

【解説】
vikThumbVertical の場合は、サムネイル画像は左上から右方向に表示されます。
vikThumbHorizontal の場合は、サムネイル画像は左上から下方向に表示されます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit6/7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikThumbVertical, ikThumbHorizontal)。

SelectTextBackColor,SelectTextColor,TextBackColor,TextColor
(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイル画像のファイル名および注釈の描画色と背景色を設定します。

【書式】
※SelectTextBackColor にて説明(その他も同様な使い方)
(1)C++Builder　*thumbcontrolname*->SelectTextBackColor [= *TColor*]
(2)Delphi　*thumbcontrolname*.SelectTextBackColor [= *TColor*]

【設定値】
SelectTextBackColor は選択されたコマのテキストの背景色。
SelectTextColor は選択されたコマのテキストの描画色。
TextBackColor はテキストの背景色。
TextColor はテキストの描画色。

【解説】
値を設定する場合は、色定数(clRed など)や RGB(Red,Green,Blue)の戻り値を与えます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

SelectTextTransparent,TextTransparent（サムネイルコントロール／カスタムプロパティ）

【機能】
サムネイル画像のファイル名および注釈を描画する時の透過を設定します。

【書式】
※SelectTextTransparent にて説明（TextTransparent も同様な使い方）
(1)C++Builder　*thumbcontrolname*->SelectTextTransparent [= *bool*]
(2)Delphi　*thumbcontrolname*.SelectTextTransparent [= *Boolean*]

【設定値】

値	説明
True	透過有効
False	透過無効

【解説】
SelectTextTransparent プロパティはコマを選択した時に有効で、TextTransparent プロパティはコマを選択していない時に有効です。
プロパティを True に設定した場合、背景色（SelectTextBackColor,TextBackColor プロパティ）は無視されます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Sort(サムネイルコントロール／カスタムプロパティ)

【機能】
FilePath プロパティに設定されたフォルダのファイルや ListOfFileNames プロパティに設定されたファイル名のリストへのソート方法を設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Sort** [= *TVIkSort*]
(2)Delphi　*thumbcontrolname*.**Sort** [= *TVIkSort*]

【TVIkSort 型】
ユニット
IkInit
type
TVIkSort = (vikSortNone, vikSortFileName, vikSortDate, vikSortFileExtName, vikSortFileExtDate, vikSortDirFileName, vikSortDirDate, vikSortDirFileExtName, vikSortDirFileExtDate);

【設定値】

値	説明
vikSortNone	しない
vikSortFileName	ファイル名
vikSortDate	日付
vikSortFileExtName	ファイル拡張子＋ファイル名
vikSortFileExtDate	ファイル拡張子＋日付
vikSortDirFileName	フォルダ名＋ファイル名
vikSortDirDate	フォルダ名＋日付
vikSortDirFileExtName	フォルダ名＋ファイル拡張子＋ファイル名
vikSortDirFileExtDate	フォルダ名＋ファイル拡張子＋日付

【解説】
日付は SortDate プロパティに設定された日付が使用されます。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit5 ActiveX との違い】
Sort プロパティに含まれる日付は更新日時を表していました。SortDate プロパティに vikSortDateLastWriteTime を設定すると IK5 と同じ動作になります。

【ImageKit6 ActiveX との違い】
・列挙型の識別子の先頭に v が付加されました(ActiveX は ikSortNone, ikSortFileName, ikSortDate, ikSortFileExtName, ikSortFileExtDate, ikSortDirFileName, ikSortDirDate, ikSortDirFileExtName, ikSortDirFileExtDate)。
・Sort プロパティに含まれる日付は更新日時を表していました。SortDate プロパティに vikSortDateLastWriteTime を設定すると IK6 と同じ動作になります。

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikSortNone, ikSortFileName, ikSortDate, ikSortFileExtName, ikSortFileExtDate, ikSortDirFileName, ikSortDirDate, ikSortDirFileExtName, ikSortDirFileExtDate)。

【ImageKit6 VCL との違い】
Sort プロパティに含まれる日付は更新日時を表していました。SortDate プロパティに vikSortDateLastWriteTime を設定すると IK6 と同じ動作になります。

SortDate(サムネイルコントロール／カスタムプロパティ)

【機能】
Sort プロパティに日付が含まれる場合のソートする日付を設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**SortDate** [= *TVIkSortDate*]
(2)Delphi　*thumbcontrolname*.**SortDate** [= *TVIkSortDate*]

【TVIkSortDate 型】
ユニット
IkInit
type
TVIkSortDate = (vikSortDateCreationTime, vikSortDateLastWriteTime, vikSortDateLastAccessTime);

【設定値】

値	説明
vikSortDateCreationTime	作成日時
vikSortDateLastWriteTime	更新日時
vikSortDateLastAccessTime	アクセス日時

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikSortDateCreationTime, ikSortDateLastWriteTime, ikSortDateLastAccessTime)。

SortOrder(サムネイルコントロール／カスタムプロパティ)

【機能】
FilePath プロパティに設定されたフォルダのファイルや ListOfFileNames プロパティに設定されたファイル名のリストへのソート順を設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**SortOrder** [= *TVIkSortOrder*]
(2)Delphi　*thumbcontrolname*.**SortOrder** [= *TVIkSortOrder*]

【TVIkSortOrder 型】
ユニット
IkInit
type
TVIkSortOrder = (vikSortOrderInc, vikSortOrderDec);

【設定値】

値	説明
vikSortOrderInc	昇順
vikSortOrderDec	降順

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit6/7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikSortOrderInc, ikSortOrderDec)。

StartNum(サムネイルコントロール／カスタムプロパティ)

【機能】
表示を開始するイメージの番号を取得もしくは設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**StartNum** [= *short*]
(2)Delphi　*thumbcontrolname*.**StartNum** [= *Smallint*]

【設定値】
表示を開始するイメージの番号(0～)

【解説】
サムネイルコントロールに表示している左上のサムネイル画像の番号です。
イメージの番号は 0 から始まり、順番はソートなどの条件に依存します。
ColCount プロパティの値や **RowCount** プロパティの値で割り切れない番号を設定すると、割り切れる番号に再設定します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Style(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイル画像を表示する背景のデザインを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Style** [= *TVIkThumbnailStyle*]
(2)Delphi　*thumbcontrolname*.**Style** [= *TVIkThumbnailStyle*]

【TVIkThumbnailStyle 型】
ユニット
IkThumb
type
TVIkThumbnailStyle = (vikButton, vikFilm, vikCustom);

【設定値】

値	説明
vikButton	ボタン
vikFilm	フィルム
vikCustom	カスタム

【解説】
実行時にデザインを変更する場合には、再度 **Display** メソッドを実行する必要があります。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit6/7/8/9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikButton, ikFilm, ikCustom)。

Text(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイル画像の注釈を設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Text** [= *UnicodeString*]
(2)Delphi　*thumbcontrolname*.**Text** [= *string*]

【設定値】
注釈。

【解説】
注釈を設定する場合は **ShowThumbImage** イベント内で行ってください。
Text プロパティを有効にするには **DispFileName** プロパティを True に設定してください。
設定された注釈はサムネイル画像の下に表示されます。

【値の設定】 実行時

【値の参照】 不可

ThumbArrayFileName,ThumbArrayPage,ThumbArrayPathName
(サムネイルコントロール／カスタムプロパティ)

【機能】
ThumbArrayNum プロパティで指定したサムネイル画像のフォルダ名、ファイル名、ページ番号を取得します。

【書式】
※ThumbArrayFileName にて説明(ThumbArrayPathName も同様な使い方)
(1)C++Builder　*thumbcontrolname*->ThumbArrayFileName [= *UnicodeString*]
(2)Delphi　*thumbcontrolname*.ThumbArrayFileName [= *string*]

※ThumbArrayPage
(1)C++Builder　*thumbcontrolname*->ThumbArrayPage [= *short*]
(2)Delphi　*thumbcontrolname*.ThumbArrayPage [= *Smallint*]

【参照値】
ThumbArrayFileName はファイル名。
ThumbArrayPage はファイルのページ番号(0～)。
ThumbArrayPathName はフォルダ名。

【解説】
ThumbArrayPage プロパティはマルチイメージファイルの場合に参照します。(シングルイメージの場合は常に 0 となります。)
ThumbArrayNum プロパティで示されるサムネイル画像がファイルからロードされたものでない場合は、ThumbArrayFileName プロパティには一連番号、ThumbArrayPathName プロパティには空文字列、ThumbArrayPage プロパティには 0 が設定されます。

【値の設定】 不可

【値の参照】 実行時

ThumbArrayNum(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイル画像の一連番号からファイル名やフォルダ名を取得する際に使用します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ThumbArrayNum** [= *short*]
(2)Delphi　*thumbcontrolname*.**ThumbArrayNum** [= *Smallint*]

【設定値】
サムネイル画像の一連番号(1～)

【解説】
当プロパティにイメージ番号を設定し、ThumbArrayFileName,ThumbArrayPage,ThumbArrayPathName プロパティを参照します。

【値の設定】 実行時

【値の参照】 不可

ThumbnailFile(サムネイルコントロール／カスタムプロパティ)

【機能】
サムネイルコントロールで使用するサムネイルファイルを設定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**ThumbnailFile** [= *UnicodeString*]
(2)Delphi　*thumbcontrolname*.**ThumbnailFile** [= *string*]

【設定値】
ファイル名(フルパス)。

【解説】
ファイル名を設定すると、2 回目以降の読み込みが速くなります。**FilePath** プロパティに設定するフォルダ毎に用意し、**FilePath** プロパティを新たに設定する場合は一緒に変更するようにしてください。
ListOfFileNames プロパティに設定したイメージファイルのリストのみを表示する場合(サムネイルファイルに含まれる画像は表示しない)は当プロパティに空文字列を設定してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Touch(サムネイルコントロール／カスタムプロパティ)

【機能】
タッチスクリーン使用時に指によるサムネイルのスクロールを有効にするかどうかを指定します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Touch** [= *TTouchManager**]
(2)Delphi　*thumbcontrolname*.**Touch** [= *TTouchManager*]

【設定値】
指によるスクロールを有効にする場合は **Touch.InteractiveGestures** に **igPan** を設定してください。
指によるスクロールを無効にする場合は **Touch.InteractiveGestures** に **igPan** を設定しないでください。

【解説】
タッチスクリーン対応アプリケーションで有効な機能です。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
プロパティの名称が **EnbaleTouch** から変更されました。

Clear(サムネイルコントロール／カスタムメソッド)

【機能】
サムネイル画像の表示をクリアします。

【書式】
(1)C++Builder　*thumbcontrolname*->**Clear**()
(2)Delphi　*thumbcontrolname*.**Clear**

【引数】
ありません。

【戻り値】
ありません。

Delete（サムネイルコントロール／カスタムメソッド）

【機能】
指定したサムネイル画像をサムネイルコントロールから削除します。

【書式】
(1)C++Builder　[*bool =*]*thumbcontrolname*->**Delete**(short ImageNumber)
(2)Delphi　[*Boolean =*]*thumbcontrolname*.**Delete**(ImageNumber: Smallint)

【引数】

名称	内容
ImageNumber	サムネイルの一連番号（1～）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
サムネイルコントロールからサムネイル画像を削除しますが、実ファイルは削除しません。
指定したサムネイル画像を次から表示したくない場合は（再度ロードした場合など）、メソッド実行後（戻り値が True の時）に実ファイルを削除してください。
実ファイルの名称はメソッド実行前に **ThumbArrayNum** プロパティにサムネイルの一連番号を設定することにより **ThumbArrayFileName**,**ThumbArrayPage**,**ThumbArrayPathName** プロパティに設定されます。

DeSelectImage（サムネイルコントロール／カスタムメソッド）

【機能】
指定したサムネイル画像の選択を解除します。

【書式】
(1)C++Builder　[*bool* =]*thumbcontrolname*->**DeSelectImage**(short ImageNumber)
(2)Delphi　[*Boolean* =]*thumbcontrolname*.**DeSelectImage**(ImageNumber: Smallint)

【引数】

名称	内容
ImageNumber	サムネイルの一連番号（1～）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
指定したサムネイル画像が選択されていない場合は、何も行いません。
引数の ImageNumber を **ThumbArrayNum** プロパティに設定することにより、**ThumbArrayFileName**,**ThumbArrayPage**, **ThumbArrayPathName** プロパティを参照できます。

Display（サムネイルコントロール／カスタムメソッド）

【機能】
サムネイル画像を表示します。

【書式】
(1)C++Builder　[*bool =*]*thumbcontrolname*->**Display**()
(2)Delphi　[*Boolean =*]*thumbcontrolname*.**Display**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドを実行する場合は、予め **GetFiles** メソッドを実行して表示するファイルの情報を取得する必要があります。また、**ColCount**、**RowCount** プロパティが設定されている必要があります。

GetFiles（サムネイルコントロール／カスタムメソッド）

【機能】
サムネイル画像を取得します。

【書式】
(1)C++Builder　[*bool =*]*thumbcontrolname*->**GetFiles**()
(2)Delphi　[*Boolean =*]*thumbcontrolname*.**GetFiles**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドを実行する場合は、予め **ReadMode** プロパティと下記のプロパティを設定する必要があります。

(1)フォルダ名とファイルの拡張子を指定する場合
FilePath、**FileExtension** プロパティに有効な値を設定
(2)ファイル名を指定する場合
ListOfFileNames プロパティにファイル名のリストを設定し、**FilePath** プロパティに空文字列を設定

また、サムネイルファイル(**ThumbnailFile** プロパティ)が存在しない場合や存在しても使用せずにサムネイル画像を取得する場合は **Sort**、**SortDate**、**SortOrder** プロパティを使用してソートを行うことができます。

IsSelected(サムネイルコントロール／カスタムメソッド)

【機能】
指定したサムネイル画像が選択されているかどうかを判定します。

【書式】
(1)C++Builder　[*bool* =]*thumbcontrolname*->**IsSelected**(short ImageNumber)
(2)Delphi　[*Boolean* =]*thumbcontrolname*.**IsSelected**(ImageNumber: Smallint)

【引数】

名称	内容
ImageNumber	サムネイルの一連番号(1～)

【戻り値】
選択されている場合は True、選択されていない場合もしくは失敗した場合は False を返します。

【解説】
マウスやキーボード、または **SelectImage** メソッドで選択したサムネイル画像の情報を取得する際に使用します。
引数の ImageNumber を **ThumbArrayNum** プロパティに設定することにより、**ThumbArrayFileName**,**ThumbArrayPage**, **ThumbArrayPathName** プロパティを参照できます。

Refresh（サムネイルコントロール／カスタムメソッド）

【機能】
サムネイルコントロールのイメージの再描画を実行します。

【書式】
(1)C++Builder　*thumbcontrolname*->**Refresh**()
(2)Delphi　*thumbcontrolname*.**Refresh**

【引数】
ありません。

【戻り値】
ありません。

Scroll(サムネイルコントロール／カスタムメソッド)

【機能】

ScrollStep プロパティで設定した行または列数を、ScrollVH プロパティで設定した方向にサムネイル画像をスクロールさせます。

【書式】

(1)C++Builder　[*bool =*]*thumbcontrolname*->**Scroll**()
(2)Delphi　[*Boolean =*]*thumbcontrolname*.**Scroll**

【引数】

ありません。

【戻り値】

成功の場合は True、失敗の場合は False を返します。

【解説】

このメソッドを実行する場合は、予め ScrollStep、ScrollVH プロパティに適切な値が設定されている必要があります。

SelectImage（サムネイルコントロール／カスタムメソッド）

【機能】
指定したサムネイル画像を選択します。

【書式】
(1)C++Builder　[*bool =*]*thumbcontrolname*->**SelectImage**(short ImageNumber)
(2)Delphi　[*Boolean =*]*thumbcontrolname*.**SelectImage**(ImageNumber: Smallint)

【引数】

名称	内容
ImageNumber	サムネイルの一連番号（1～）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
動作は、マウスクリックやキーボードと同様です。**EnableSelectMultiFiles** プロパティが False の場合、最後に選択されたサムネイル画像が有効となります（それ以外は解除されます）。
引数の ImageNumber を **ThumbArrayNum** プロパティに設定することにより、**ThumbArrayFileName**,**ThumbArrayPage**, **ThumbArrayPathName** プロパティを参照できます。

SortImage(サムネイルコントロール／カスタムメソッド)

【機能】
サムネイル画像をソートします。

【書式】
(1)C++Builder　[*bool =*]*thumbcontrolname*->**SortImage**()
(2)Delphi　[*Boolean =*]*thumbcontrolname*.**SortImage**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドを実行する場合は、予め **Sort**,**SortDate**,**SortOrder** プロパティに適切な値が設定されている必要があります。
たとえば、エクスプローラから画像ファイルをサムネイルコントロール上にドラッグ＆ドロップし、その画像をソートしたい場合などに使用します(ファイルからの読み込みは最初の **GetFiles** メソッドでソートが可能)。
事前にサムネイル画像などを移動させて順番を変更している場合やサムネイルファイル(**ThumbnailFile** プロパティ)から画像を取得している場合はソートを行うことにより順番が入れ替わる可能性がありますので注意してください。

UpdateThumbnailFile（サムネイルコントロール／カスタムメソッド）

【機能】
サムネイルファイルを更新します。

【書式】
(1)C++Builder　*thumbcontrolname*->**UpdateThumbnailFile**()
(2)Delphi　*thumbcontrolname*.**UpdateThumbnailFile**

【引数】
ありません。

【戻り値】
ありません。

【解説】
ThumbnailFile プロパティで設定されたサムネイルファイルを更新します。
FilePath プロパティを切り替えて別のフォルダのサムネイル画像を表示する場合や、サムネイルの処理を終了する場合などに実行してください。なお、**ImageHandle** プロパティに設定したサムネイル画像はサムネイルファイルには保存されません。

DropFileInThumb（サムネイルコントロール／カスタムイベント）

【機能】
サムネイルコントロールにファイルをドロップすると発生します。

【書式】
(1)C++Builder
*thumbcontrolname*DropFileInThumb(TObject *Sender, const UnicodeString PathName, const UnicodeString FileName)
(2)Delphi
*thumbcontrolname*DropFileInThumb(Sender: TObject; const PathName, FileName: string)

【引数】

名称	内容
PathName	ドロップするイメージのフォルダ名
FileName	ドロップするイメージのファイル名

【解説】
EnableDragTarget プロパティが True の場合に有効となります。

【ImageKit6/7/8/9/10 ActiveX との違い】
・C++Builder で使用する場合、引数の PathName と FileName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の PathName と FileName が string 型に変更されました。

EndLoadFile(サムネイルコントロール／カスタムイベント)

【機能】
サムネイルコントロール上に 1 イメージをロードした後に発生します。

【書式】
(1)C++Builder
*thumbcontrolname***EndLoadFile**(TObject *Sender, const UnicodeString PathName, const UnicodeString FileName, short ImageNumber, short MaxImage)
(2)Delphi
*thumbcontrolname***EndLoadFile**(Sender: TObject; const PathName, FileName: string; ImageNumber, MaxImage: Smallint)

【引数】

名称	内容
PathName	ロードするイメージのフォルダ名
FileName	ロードするイメージのファイル名
ImageNumber	サムネイルの一連番号
MaxImage	イメージの総数

【解説】
実画像をロードした後に発生します。ただし、サムネイルファイル(**ThumbnailFile** プロパティ)からロードする時は当イベントは発生しません。
実画像のロードに失敗した場合は、ImageNumber は 0 となります。
処理を中止する場合は **Cancel** プロパティを True に設定します。

マルチイメージファイルの場合は、FileName にページ番号が付加されます。
(ただし、ページ番号が 0 の場合は除きます。)
例えば、ABC.TIF ファイルのページ番号が 2 の場合は FileName に"ABC.TIF 2"が設定されます。

【ImageKit5/6/7/8/9/10 ActiveX との違い】
・C++Builder で使用する場合、引数の PathName と FileName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の PathName と FileName が string 型に変更されました。

MouseMoveOnThumb(サムネイルコントロール／カスタムイベント)

【機能】
サムネイルコントロール上のコマ(サムネイル画像を表示する領域)をマウスで移動した時に発生します。

【書式】
(1)C++Builder
*thumbcontrolname*MouseMoveOnThumb(TObject *Sender, const UnicodeString PathName, const UnicodeString FileName, short ImageNumber, short MaxImage, short ALeft, short ATop, short ARight, short ABottom)
(2)Delphi
*thumbcontrolname*MouseMoveOnThumb(Sender: TObject; const PathName, FileName: string; ImageNumber, MaxImage, ALeft, ATop, ARight, ABottom: Smallint)

【引数】

名称	内容
PathName	移動先のイメージのフォルダ名
FileName	移動先のイメージのファイル名
ImageNumber	サムネイルの一連番号
MaxImage	イメージの総数
ALeft	移動先のサムネイル画像が属するコマの左位置(ピクセル)
ATop	移動先のサムネイル画像が属するコマの上位置(ピクセル)
ARight	移動先のサムネイル画像が属するコマの右位置(ピクセル)
ABottom	移動先のサムネイル画像が属するコマの下位置(ピクセル)

【解説】
サムネイルコントロールにサムネイル画像が全く表示されていない場合、イベントは発生しません。また、移動先のコマに画像が表示されていない場合、PathName と FileName には空文字列が設定されます。

マルチイメージファイルの場合は、FileName にページ番号が付加されます。
(ただし、ページ番号が 0 の場合は除きます。)
例えば、ABC.TIF ファイルのページ番号が 2 の場合は FileName に"ABC.TIF 2"が設定されます。

【ImageKit6/7/8/9/10 ActiveX との違い】
・C++Builder で使用する場合、引数の PathName と FileName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の PathName と FileName が string 型に変更されました。
・**EnableMouseMoveButton** プロパティの値に関わらずマウス移動時にイベントが発生します(**ImageKit10 は除く**)。

【ImageKit6/7/8/9 VCL との違い】
EnableMouseMoveButton プロパティの値に関わらずマウス移動時にイベントが発生します。

SelectFile(サムネイルコントロール／カスタムイベント)

【機能】
サムネイルコントロール上のコマ(サムネイル画像を表示する領域)を選択すると発生します。

【書式】
(1)C++Builder
*thumbcontrolname*SelectFile(TObject *Sender, const UnicodeString PathName, const UnicodeString FileName, short ImageNumber, short MaxImage, short ALeft, short ATop, short ARight, short ABottom)
(2)Delphi
*thumbcontrolname*SelectFile(Sender: TObject; const PathName, FileName: string; ImageNumber, MaxImage, ALeft, ATop, ARight, ABottom: Smallint)

【引数】

名称	内容
PathName	選択されたイメージのフォルダ名
FileName	選択されたイメージのファイル名
ImageNumber	サムネイルの一連番号
MaxImage	イメージの総数
ALeft	選択されたサムネイル画像が属するコマの左位置(ピクセル)
ATop	選択されたサムネイル画像が属するコマの上位置(ピクセル)
ARight	選択されたサムネイル画像が属するコマの右位置(ピクセル)
ABottom	選択されたサムネイル画像が属するコマの下位置(ピクセル)

【解説】
サムネイルコントロールにサムネイル画像が全く表示されていない場合、イベントは発生いたしません。また、選択されたコマに画像が表示されていない場合、PathName と FileName には空文字列が設定されます。
マルチイメージファイルの場合は、FileName にページ番号が付加されます。(ただし、ページ番号が 0 の場合は除きます。)
例えば、ABC.TIF ファイルのページ番号が 2 の場合は FileName に"ABC.TIF 2"が設定されます。

Ctrl キーを押しながらマウスをクリックすると、サムネイル画像の選択および解除(選択された画像をクリックすると解除)ができますが、いずれの場合でもイベントは発生します。該当するサムネイル画像が選択されているかどうかについては IsSelected メソッドを使用して判定してください。
Shift キーを押しながらマウスをクリックすると、先頭から最後までの該当するサムネイル画像の数だけイベントが発生します。

サムネイル画像は、マウスクリックやマウスの移動、およびキーボードや SelectImage メソッドによって選択できます。

【ImageKit5 ActiveX との違い】
・引数に ALeft,ATop,ARight,ABottom が追加されました。
・C++Builder で使用する場合、引数の PathName と FileName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の PathName と FileName が string 型に変更されました。

【ImageKit6/7/8/9/10 ActiveX との違い】
・C++Builder で使用する場合、引数の PathName と FileName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の PathName と FileName が string 型に変更されました。

ShowThumbImage(サムネイルコントロール／カスタムイベント)

【機能】
サムネイル画像を表示した後に発生します。

【書式】
(1)C++Builder
*thumbcontrolname***ShowThumbImage**(TObject *Sender, const UnicodeString PathName, const UnicodeString FileName, short ImageNumber, short MaxImage, short ALeft, short ATop, short ARight, short ABottom)
(2)Delphi
*thumbcontrolname***ShowThumbImage**(Sender: TObject; const PathName, FileName: string; ImageNumber, MaxImage, ALeft, ATop, ARight, ABottom: Smallint)

【引数】

名称	内容
PathName	表示するイメージのフォルダ名
FileName	表示するイメージのファイル名
ImageNumber	サムネイルの一連番号
MaxImage	イメージの総数
ALeft	表示するサムネイル画像が属するコマの左位置(ピクセル)
ATop	表示するサムネイル画像が属するコマの上位置(ピクセル)
ARight	表示するサムネイル画像が属するコマの右位置(ピクセル)
ABottom	表示するサムネイル画像が属するコマの下位置(ピクセル)

【解説】
マルチイメージファイルの場合は、FileName にページ番号が付加されます。
(ただし、ページ番号が 0 の場合は除きます。)
例えば、ABC.TIF ファイルのページ番号が 2 の場合は FileName に"ABC.TIF 2"が設定されます。

【ImageKit6/7/8/9/10 ActiveX との違い】
・C++Builder で使用する場合、引数の PathName と FileName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の PathName と FileName が string 型に変更されました。

StartLoadFile（サムネイルコントロール／カスタムイベント）

【機能】
サムネイルコントロール上に 1 イメージをロードする前に発生します。

【書式】
(1)C++Builder
*thumbcontrolname***StartLoadFile**(TObject *Sender, const UnicodeString PathName, const UnicodeString FileName, short ImageNumber, short MaxImage)
(2)Delphi
*thumbcontrolname***StartLoadFile**(Sender: TObject; const PathName, FileName: string; ImageNumber, MaxImage: Smallint)

【引数】

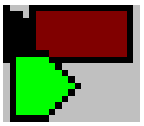
名称	内容
PathName	ロードするイメージのフォルダ名
FileName	ロードするイメージのファイル名
ImageNumber	サムネイルの一連番号
MaxImage	イメージの総数

【解説】
実画像をロードする前に発生します。ただし、サムネイルファイル（**ThumbnailFile** プロパティ）からロードする時は当イベントは発生しません。
処理を中止する場合は、**Cancel** プロパティを True に設定します。

マルチイメージファイルの場合は、FileName にページ番号が付加されます。
（ただし、ページ番号が 0 の場合は除きます。）
例えば、ABC.TIF ファイルのページ番号が 2 の場合は FileName に"ABC.TIF 2"が設定されます。

【ImageKit5/6/7/8/9/10 ActiveX との違い】
・C++Builder で使用する場合、引数の PathName と FileName が UnicodeString 型に変更されました。
・Delphi で使用する場合、引数の PathName と FileName が string 型に変更されました。

1－3. プレイコントロール



プレイコントロール
指定したファイルを開き、動画を再生します。動画の再生、停止、一時停止、静止画の取得ができます。
※Delphi 専用のコントロールです（C++Builder へは非対応）。

継承
TCustomControl
ユニット
IkPlay

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Appearance	プレイコントロールの縁の設定
BorderColor	プレイコントロールの縁の色
BorderVisible	プレイコントロールの縁の描画設定
Color	プレイコントロールの背景色
Cursor	プレイコントロールの描画領域内におけるマウスカーソルの形状
Duration	再生時間を秒単位で取得
Enabled	マウスやキーボードイベントへの応答可否
ErrorStatus	エラーの種類を示す
FrameRate	フレームレートを取得
Handle	プレイコントロールのウィンドウハンドル
State	現在の状態を取得
VideoSize	動画の幅と高さを取得

【ImageKit9/10 ActiveX との違い】
削除されたプロパティ： MouseCursorFile
※MouseCursorFile プロパティは Cursor プロパティで代用

変更されたプロパティ:
BackColor --> Color
HWND --> Handle
MouseCursorType --> Cursor
VideoHeight,VideoWidth --> VideoSize

●メソッド一覧（アルファベット順）

カスタムメソッド	内容
Close	動画ファイルを閉じる
Open	動画ファイルを開く
Pause	動画の一時停止、または一時停止した動画の再生
Start	動画の再生
StartFrom	動画の開始位置を設定
Stop	動画の停止
TakeSnapshot	静止画の取得

●イベント一覧（アルファベット順）

カスタムイベント	内容
End	再生が完了した後に発生
Snapshot	静止画を取得した後に発生

Appearance(プレイコントロール／カスタムプロパティ)

【機能】
プレイコントロールの縁の表示を設定します。

【書式】
Delphi　*playcontrolname*.**Appearance** [= *TVIkAppearance*]

【TVIkAppearance 型】
ユニット
IkInit
type
TVIkAppearance = (vikFlat, vikLowered, vikRaised);

【設定値】

値	説明
vikFlat	Flat
vikLowered	凹
vikRaised	凸

【解説】
BorderVisible プロパティが True の場合に有効です。
vikFlat の場合、**BorderColor** プロパティに設定されている色で縁を描画します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikFlat, ikLowered, ikRaised)。

BorderColor,Color(プレイコントロール／カスタムプロパティ)

【機能】
色情報を設定します。

【書式】
※BorderColor にて説明(Color も同様な使い方)
Delphi　*playcontrolname*.BorderColor [= *TColor*]

【設定値】
BorderColor はプレイコントロールの縁の色。
Color はプレイコントロールの背景色。

【解説】
値を設定する場合は、色定数(clRed など)や RGB(Red,Green,Blue)の戻り値などを与えます。
BorderColor プロパティは Appearance プロパティが vikFlat で BorderVisible プロパティが True の場合に有効です。
TColor 型については Delphi のヘルプを参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
Color プロパティの名称が BackColor から変更されました。

BorderVisible(プレイコントロール／カスタムプロパティ)

【機能】
プレイコントロールの縁を描画するかどうかを設定します。

【書式】
Delphi　*playcontrolname*.**BorderVisible** [= *Boolean*]

【設定値】

値	説明
True	縁を描画する
False	縁を描画しない

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Cursor(プレイコントロール／カスタムプロパティ)

【機能】
プレイコントロールの描画領域内におけるマウスカーソルの形状を取得または設定します。

【書式】
Delphi　*playcontrolname*.**Cursor** [= *TCursor*]

【設定値】
crDefault や crArrow など。詳しくは Delphi のヘルプを参照のこと。

【解説】
デフォルト値は crDefault です。
カスタムカーソルを割り当てる場合は、Delphi のヘルプの TControl.Cursor の例を参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
MouseCursorFile,MouseCursorType プロパティの機能が **Cursor** プロパティに統一されました。

Duration(プレイコントロール／カスタムプロパティ)

【機能】
再生時間を秒単位で取得します。

【書式】
Delphi　*playcontrolname*.**Duration** [= *Double*]

【参照値】
再生時間(秒単位)。

【解説】
Open メソッド実行後に更新されます。

【値の設定】 不可

【値の参照】 実行時

Enabled(プレイコントロール／カスタムプロパティ)

【機能】
プレイコントロールのマウスやキーボードイベントに応答するかどうかを設定します。

【書式】
Delphi　*playcontrolname*.**Enabled** [= *Boolean*]

【設定値】

値	説明
True	マウスやキーボードイベントを有効
False	マウスやキーボードイベントを無効

【解説】
True を設定するとマウスダウンやキーダウンなどのイベントに応答できますが、False の場合は該当イベントが発生しません。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ErrorStatus(プレイコントロール／カスタムプロパティ)

【機能】
プレイコントロールを使用して起きたエラーの種類を示します。

【書式】
Delphi　*playcontrolname*.**ErrorStatus** [= *Smallint*]

【参照値】

値	説明
0	正常終了(エラーなし)
2	その他のエラー
4	サポート外機能
5	設定値が不正
11	実行順序エラー
20	ファイルがオープンできない
201	DirectX の環境が整っていない
203	COM インターフェイスに関するエラー

【解説】
プレイコントロールのメソッドを実行することにより、値を参照できます。

【値の設定】 不可

【値の参照】 実行時

FrameRate(プレイコントロール／カスタムプロパティ)

【機能】
フレームレートを取得します。

【書式】
Delphi　*playcontrolname*.**FrameRate** [= *Double*]

【参照値】
フレームレート(フレーム/秒)。

【解説】
Open メソッド実行後に更新されます。

【値の設定】 不可

【値の参照】 実行時

Handle(プレイコントロール／カスタムプロパティ)

【機能】
プレイコントロールのウィンドウハンドルを示します。

【書式】
Delphi　*playcontrolname*.**Handle** [= *HWND*]

【参照値】
プレイコントロールのウィンドウハンドル。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
プロパティの名称が **HWND** から変更されました。

State(プレイコントロール／カスタムプロパティ)

【機能】
現在の状態を取得します。

【書式】
Delphi *playcontrolname*.**State** [= *TVIkVideoStates*]

【TVIkVideoStates 型】
ユニット
IkWebCamera
type
TVIkVideoState = (vikClosed = 0, vikOpen = 1, vikRunning = 2, vikPaused = 4, vikStopped = 8);
TVIkVideoStates = set of TVIkVideoState;

【参照値】

値	説明
vikClosed	接続を閉じた、もしくは接続されていない
vikOpen	接続しています
vikRunning	実行中
vikPaused	一時停止しました
vikStopped	停止しました

【解説】
Close,**Open**,**Pause**,**Start**,**Stop** メソッド実行後に更新されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
集合型に変更されました(ActiveX は ikClosed, ikOpen, ikRunning, ikPaused, ikStopped。複数の組み合わせの場合、論理和となる)。

VideoSize(プレイコントロール／カスタムプロパティ)

【機能】
動画の高さと幅を取得します。

【書式】
Delphi　*playcontrolname*.**VideoSize** [= *TSize*]

【参照値】
VideoSize.cx は動画の幅。
VideoSize.cy は動画の高さ。

【解説】
Open メソッド実行後に更新されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
VideoHeight,VideoWidth プロパティの機能が **VideoSize** プロパティに統一されました。

Close(プレイコントロール／カスタムメソッド)

【機能】
動画ファイルを閉じます。

【書式】
Delphi ［ *Boolean* =］*playcontrolname*.**Close**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Open** メソッドで開いたファイルを閉じます。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 11 が設定されます。**Open** メソッドを実行せずに **Close** メソッドを実行した場合が該当します。

Open(プレイコントロール／カスタムメソッド)

【機能】
動画ファイルを開きます。

【書式】
Delphi [*Boolean* =]*playcontrolname*.**Open**(const FileName: string)

【引数】

名称	内容
FileName	ファイルの名称

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数の FileName には再生する動画ファイルの名称を渡します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 11,20,201,203 のいずれかが設定されます。

注意事項
プレイコントロールは DirectShow の機能を利用しておりますが、標準では一般的なメディア形式のフィルタしか導入されないため、対応していないコーデックがあります。それに該当する形式のファイルを開く場合は別途コーデックを入手する必要があります。

Pause(プレイコントロール／カスタムメソッド)

【機能】
動画を一時停止、または一時停止した動画を再生します。

【書式】
Delphi ［ *Boolean* =］*playcontrolname*.**Pause**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Start** メソッドで開始した動画を一時停止します。また、一時停止した状態で再度このメソッドを実行すると動画を再生します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2 や 11 が設定されます。

Start(プレイコントロール／カスタムメソッド)

【機能】
動画を再生します。

【書式】
Delphi [*Boolean* =]*playcontrolname*.**Start**(Mode: TVIkVideoDisplayMode; Rate: Double)

【TVIkVideoDisplayMode 型】
ユニット
IkWebCamera
type
TVIkVideoDisplayMode = (vikVideoScale = 0, vikVideoStretch = 1);

【引数】

名称	内容
Mode	表示モード vikVideoScale:スケール(映像とコントロールの縦横比を保持して表示) vikVideoStretch:ストレッチ(映像をコントロールの縦横サイズに合わせて表示)
Rate	速度

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数の Mode には映像をどのように表示するかを渡します。
引数の Rate には再生速度を渡します。通常は 1.0。0 または負の数は不可です。また、再生するファイル(たとえば WMV 形式など)により Rate が有効にならない場合があります(常に 1.0)。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2,5,11 のいずれかが設定されます。
再生開始位置を設定する場合は、事前に **StartFrom** メソッドを実行してください。

StartFrom(プレイコントロール／カスタムメソッド)

【機能】
動画の再生開始位置を設定します。

【書式】
Delphi
[*Boolean =*]*playcontrolname*.**StartFrom**(Time: Double)
[*Boolean =*]*playcontrolname*.**StartFrom**(Time: Int64)

【引数】

名称	内容
Time	時間(引数が Double 型の場合は秒単位、Int64 型の場合は 100 ナノ秒単位です)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数の Time には再生開始位置を渡します。通常は 0。負の数は不可です。
戻り値が False の場合、**ErrorStatus** プロパティには 2,5,11 のいずれかが設定されます。
Open メソッド実行直後は **StartFrom** メソッドを実行せずに **Start** メソッドを実行しても最初から再生されますが、同じ動画を繰り返し再生する場合は **Start** メソッドを実行する前に **StartFrom** メソッドで再生開始位置を設定してください。

Stop(プレイコントロール／カスタムメソッド)

【機能】
動画を停止します。

【書式】
Delphi ［ *Boolean* =］*playcontrolname*.**Stop**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Start** メソッドで再生した動画を停止します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 11 が設定されます。

TakeSnapshot(プレイコントロール／カスタムメソッド)

【機能】
静止画を取得します。

【書式】
Delphi ［ *Boolean* =］*playcontrolname*.**TakeSnapshot**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Start** メソッドで再生した動画から静止画を取得します。静止画は **Snapshot** イベントで取得できます。
戻り値が False の場合、**ErrorStatus** プロパティには 2,4,11 のいずれかが設定されます。4 の場合、再生した動画から静止画を取得することはできません。

End(プレイコントロール／カスタムイベント)

【機能】
再生が完了した後に発生します。

【書式】
Delphi　*playcontrolname*.**End**(Sender: TObject)

【引数】
ありません。

【解説】
Start メソッドを実行して、**Stop** メソッドを実行せずに再生が完了した場合にイベントが発生します。

Snapshot（プレイコントロール／カスタムイベント）

【機能】

TakeSnapshot メソッドが成功すると発生します。

【書式】

Delphi　*playcontrolname*.**Snapshot**(Sender: TObject; ImageHandle: THandle; var FreeMemory: Boolean)

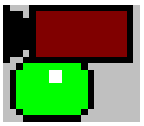
【引数】

名称	内容
ImageHandle	静止画のビットマップを示すメモリハンドル
FreeMemory	ImageHandle をイベント終了後に解放するかどうかを設定。既定値は True。 解放する場合は True、解放しない場合は False を設定します。

【解説】

引数の FreeMemory が True の場合は、取得した静止画のビットマップはイベント終了後に解放されますので、ImageHandle を残しておきたい場合は、イメージキットコントロールの **CopyImage** メソッドを実行するかファイルなどに保存してください。

1－4. プレビューコントロール



プレビューコントロール
Web(USB)カメラと接続して、映像を表示します。
表示の開始、停止、一時停止、静止画の取得ができます。
※Delphi 専用のコントロールです（C++Builder へは非対応）。

継承
TCustomControl
ユニット
IkPreview

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Appearance	プレビューコントロールの縁の設定
BitCount	プレビュー画像の 1 ピクセルあたりのビット数
BorderColor	プレビューコントロールの縁の色
BorderVisible	プレビューコントロールの縁の描画設定
Color	プレビューコントロールの背景色
Compression	プレビュー画像の圧縮フォーマット
Cursor	プレビューコントロールの描画領域内におけるマウスカーソルの形状
Enabled	マウスやキーボードイベントへの応答可否
ErrorStatus	エラーの種類を示す
FrameRate	フレームレートを取得
Handle	プレビューコントロールのウィンドウハンドル
MaximumFrameRate	最大のフレームレート
MinimumFrameRate	最小のフレームレート
State	現在の状態を取得
VideoDevices	ビデオデバイスの名称
VideoSize	プレビュー画像の幅と高さを取得
VideoSubType	メディアサンプルのサブタイプを指定する GUID

【ImageKit9/10 ActiveX との違い】
削除されたプロパティ： MouseCursorFile
※MouseCursorFile プロパティは Cursor プロパティで代用

変更されたプロパティ:
BackColor --> Color
HWND --> Handle
MouseCursorType --> Cursor
VideoHeight,VideoWidth --> VideoSize

●メソッド一覧（アルファベット順）

カスタムメソッド	内容
Close	Web(USB)カメラとの接続を閉じる
GetStreamFormat	指定したフォーマット機能を取得
GetStreamFormats	フォーマット機能セットを取得
GetVideoDevices	ビデオデバイスの名称を取得
Open	プレビューモードで Web(USB)カメラと接続
Pause	プレビューの一時停止、または一時停止したプレビューの再開
SetStreamFormat	指定したフォーマット機能を設定
ShowCapturePinDialog	キャプチャーピンダイアログの表示
ShowFilterDialog	フィルタダイアログの表示
Start	プレビューの開始
Stop	プレビューの停止

TakeSnapshot	静止画の取得

●イベント一覧（アルファベット順）

カスタムイベント	内容
Snapshot	静止画を取得した後に発生

Appearance(プレビューコントロール／カスタムプロパティ)

【機能】
プレビューコントロールの縁の表示を設定します。

【書式】
Delphi　*previewcontrolname*.**Appearance** [= *TVIkAppearance*]

【TVIkAppearance 型】
ユニット
IkInit
type
TVIkAppearance = (vikFlat, vikLowered, vikRaised);

【設定値】

値	説明
vikFlat	Flat
vikLowered	凹
vikRaised	凸

【解説】
BorderVisible プロパティが True の場合に有効です。
vikFlat の場合、**BorderColor** プロパティに設定されている色で縁を描画します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikFlat, ikLowered, ikRaised)。

BitCount(プレビューコントロール／カスタムプロパティ)

【機能】
プレビュー画像の 1 ピクセルあたりのビット数を示します。

【書式】
Delphi　*previewcontrolname*.**BitCount** [= *Word*]

【参照値】
1 ピクセルあたりのビット数。

【解説】
GetStreamFormat,**GetStreamFormats**,**Open**,**SetStreamFormat**,**ShowCapturePinDialog** メソッドを実行することにより、値が更新されます。

【値の設定】 不可

【値の参照】 実行時

BorderColor,Color(プレビューコントロール／カスタムプロパティ)

【機能】
色情報を設定します。

【書式】
※BorderColor にて説明(Color も同様な使い方)
Delphi　*previewcontrolname*.BorderColor [= *TColor*]

【設定値】
BorderColor はプレビューコントロールの縁の色。
Color はプレビューコントロールの背景色。

【解説】
値を設定する場合は、色定数(clRed など)や RGB(Red,Green,Blue)の戻り値などを与えます。
BorderColor プロパティは Appearance プロパティが vikFlat で BorderVisible プロパティが True の場合に有効です。
TColor 型については Delphi のヘルプを参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
Color プロパティの名称が BackColor から変更されました。

BorderVisible(プレビューコントロール／カスタムプロパティ)

【機能】
プレビューコントロールの縁を描画するかどうかを設定します。

【書式】
Delphi *previewbcontrolname*.**BorderVisible** [= *Boolean*]

【設定値】

値	説明
True	縁を描画する
False	縁を描画しない

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Compression(プレビューコントロール／カスタムプロパティ)

【機能】
プレビュー画像の圧縮フォーマットを示します。

【書式】
Delphi *previewbcontrolname*.**Compression** [= *DWord*]

【参照値】
圧縮フォーマット。

【解説】
GetStreamFormat,**GetStreamFormats**,**Open**,**SetStreamFormat**,**ShowCapturePinDialog** メソッドを実行することにより、値が更新されます。
画像が非圧縮の場合は 0(BI_RGB)や 3(BI_BITFIELDS)が設定されます。()内の説明は Windows API で使用する定数と同じ意味です。
画像が圧縮されている場合は、**FOURCC** コードが割り当てられます。

【値の設定】 不可

【値の参照】 実行時

Cursor(プレビューコントロール／カスタムプロパティ)

【機能】
プレビューコントロールの描画領域内におけるマウスカーソルの形状を取得または設定します。

【書式】
Delphi　*previewcontrolname*.**Cursor** [= *TCursor*]

【設定値】
crDefault や crArrow など。詳しくは Delphi のヘルプを参照のこと。

【解説】
デフォルト値は crDefault です。
カスタムカーソルを割り当てる場合は、Delphi のヘルプの TControl.Cursor の例を参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
MouseCursorFile,MouseCursorType プロパティの機能が **Cursor** プロパティに統一されました。

Enabled(プレビューコントロール／カスタムプロパティ)

【機能】
プレビューコントロールのマウスやキーボードイベントに応答するかどうかを設定します。

【書式】
Delphi　*previewcontrolname*.**Enabled** [= *Boolean*]

【設定値】

値	説明
True	マウスやキーボードイベントを有効
False	マウスやキーボードイベントを無効

【解説】
True を設定するとマウスダウンやキーダウンなどのイベントに応答できますが、False の場合は該当イベントが発生しません。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ErrorStatus(プレビューコントロール／カスタムプロパティ)

【機能】
プレビューコントロールを使用して起きたエラーの種類を示します。

【書式】
Delphi *previewcontrolname*.**ErrorStatus** [= *Smallint*]

【参照値】

値	説明
0	正常終了(エラーなし)
2	その他のエラー
4	サポート外機能
5	設定値が不正
11	実行順序エラー
201	DirectX の環境が整っていない
202	デバイスが接続されていない
203	COM インターフェイスに関するエラー

【解説】
プレビューコントロールのメソッドを実行することにより、値を参照できます。

【値の設定】 不可

【値の参照】 実行時

FrameRate(プレビューコントロール／カスタムプロパティ)

【機能】
フレームレートを取得または設定します。

【書式】
Delphi *previewcontrolname*.**FrameRate** [= *Double*]

【設定値】
フレームレート(フレーム/秒)。

【解説】
GetStreamFormat,**GetStreamFormats**,**Open**,**ShowCapturePinDialog** メソッドを実行することにより、値が更新されます。
また、**SetStreamFormat** メソッドを実行する前に値を設定すると、その値が有効になります。

【値の設定】 実行時

【値の参照】 実行時

Handle(プレビューコントロール／カスタムプロパティ)

【機能】
プレビューコントロールのウィンドウハンドルを示します。

【書式】
Delphi　*previewcontrolname*.**Handle** [= *HWND*]

【参照値】
プレビューコントロールのウィンドウハンドル。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
プロパティの名称が **HWND** から変更されました。

MaximumFrameRate,MinimumFrameRate(プレビューコントロール／カスタムプロパティ)

【機能】
最大/最小のフレームレートを取得します。

【書式】
※MaximumFrameRate にて説明(MinimumFrameRate も同様な使い方)
Delphi *previewcontrolname*.MaximumFrameRate [= *Double*]

【参照値】
フレームレート(フレーム/秒)。

【解説】
GetStreamFormat メソッドを実行することにより、値が更新されます。
FrameRate プロパティに値を設定する場合の目安となります。

【値の設定】 実行時

【値の参照】 実行時

State(プレビューコントロール／カスタムプロパティ)

【機能】
現在の状態を取得します。

【書式】
Delphi *previewcontrolname*.**State** [= *TVIkVideoStates*]

【TVIkVideoStates 型】
ユニット
IkWebCamera
type
TVIkVideoState = (vikClosed = 0, vikOpen = 1, vikRunning = 2, vikPaused = 4, vikStopped = 8);
TVIkVideoStates = set of TVIkVideoState;

【参照値】

値	説明
vikClosed	接続を閉じた、もしくは接続されていない
vikOpen	接続しています
vikRunning	実行中
vikPaused	一時停止しました
vikStopped	停止しました

【解説】
Close,**Open**,**Pause**,**Start**,**Stop** メソッド実行後に更新されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
集合型に変更されました(ActiveX は ikClosed, ikOpen, ikRunning, ikPaused, ikStopped。複数の組み合わせの場合、論理和となる)。

VideoDevices(プレビューコントロール／カスタムプロパティ)

【機能】
ビデオデバイスの名称を取得します。

【書式】
Delphi　*previewcontrolname*.**VideoDevices** [= *TStrings*]

【参照値】
ビデオデバイスの名称。

【解説】
GetVideoDevices メソッドが成功した場合にビデオデバイスの名称が設定されます。
接続されているビデオデバイスの数は **VideoDevices.Count** で取得できます。
TStrings については Delphi のヘルプを参照してください。

【値の設定】 不可

【値の参照】 実行時

VideoSize(プレビューコントロール／カスタムプロパティ)

【機能】
プレビュー画像の高さと幅を取得します。

【書式】
Delphi　*previewcontrolname*.**VideoSize** [= *TSize*]

【参照値】
VideoSize.cx はプレビュー画像の幅。
VideoSize.cy はプレビュー画像の高さ。

【解説】
GetStreamFormat,**GetStreamFormats**,**Open**,**SetStreamFormat**,**ShowCapturePinDialog** メソッドを実行することにより、値が更新されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
VideoHeight,**VideoWidth** プロパティの機能が **VideoSize** プロパティに統一されました。

VideoSubType(プレビューコントロール／カスタムプロパティ)

【機能】
メディアサンプルのサブタイプを指定する GUID を取得します。

【書式】
Delphi　*previewcontrolname*.**VideoSubType** [= *TGUID*]

【参照値】
GUID。

【解説】
GetStreamFormat,**GetStreamFormats**,**Open**,**SetStreamFormat**,**ShowCapturePinDialog** メソッドを実行することにより、値が更新されます。
プレビュー画像が圧縮されている場合、16 進表記で最初の 8 桁が **FOURCC** コードになります。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
文字列型からレコード型に変更されました。

Close(プレビューコントロール／カスタムメソッド)

【機能】
Web(USB)カメラとの接続を閉じます。

【書式】
Delphi ［ *Boolean* =］*previewcontrolname*.**Close**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Open** メソッドで接続した Web(USB)カメラを閉じます。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 11 が設定されます。**Open** メソッドを実行せずに **Close** メソッドを実行した場合が該当します。

GetStreamFormat(プレビューコントロール／カスタムメソッド)

【機能】
指定したインデックスのフォーマット機能を取得します。

【書式】
Delphi [*Boolean* =]*previewcontrolname*.**GetStreamFormat**(Index: Integer)

【引数】

名称	内容
Index	インデックス(-1 から **GetStreamFormats** メソッドの戻り値-1 の範囲内)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数の Index に-1 を渡した場合は現在選択されているフォーマット機能を取得します。Index に 0 から **GetStreamFormats** メソッドの戻り値-1 までの値を渡した場合は、そのインデックスに見合ったフォーマット機能を取得します。
戻り値が True の場合、**BitCount**,**Compression**,**FrameRate**,**MaximumFrameRate**,**MinimumFrameRate**,**VideoSize**, **VideoSubType** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 5 や 11 が設定されます。
GetStreamFormat メソッドを実行するには、事前に **Open** と **GetStreamFormats** メソッドを実行しておかなければなりません。

GetStreamFormats(プレビューコントロール／カスタムメソッド)

【機能】
フォーマット機能セットを取得します。

【書式】
Delphi [*Integer* =]*previewcontrolname*.GetStreamFormats

【引数】
ありません。

【戻り値】
取得した数を返します。

【解説】
戻り値が 1 以上の場合、GetStreamFormat メソッドを実行することにより、対応している出力サイズなどの情報を取得できます。
戻り値が 0 の場合、ErrorStatus プロパティには 2 や 11 が設定されます。
GetStreamFormats メソッドを実行するには、事前に Open メソッドを実行しておかなければなりません。

GetVideoDevices(プレビューコントロール／カスタムメソッド)

【機能】
ビデオデバイスの名称を取得します。

【書式】
Delphi ［ *Integer =* ］*previewcontrolname*.**GetVideoDevices**

【引数】
ありません。

【戻り値】
取得した数を返します。

【解説】
戻り値が 1 以上の場合、**VideoDevices** プロパティにビデオデバイスの名称が設定されます。
戻り値が 0 の場合、**ErrorStatus** プロパティには 201 や 202 が設定されます。

Open(プレビューコントロール／カスタムメソッド)

【機能】
プレビューモードで Web(USB)カメラと接続します。

【書式】
Delphi ［ *Boolean* =］*previewcontrolname*.**Open**(Index: Integer)

【引数】

名称	内容
Index	Web(USB)カメラのインデックス番号

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数の Index には **GetVideoDevices** メソッドで取得されるビデオデバイスの名称を示すインデックス(先頭のビデオデバイスを接続する場合は 0)を渡します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 11,201,202,203 のいずれかが設定されます。

注意事項
キャプチャー(レコードコントロール)とプレビューを同時に行うことはできません。プレビューを行う場合はレコードコントロールの **Close** メソッドを実行して一旦 Web(USB)カメラとの接続を閉じ、再度プレビューコントロールの **Open** メソッドを実行して接続してください。

Pause(プレビューコントロール／カスタムメソッド)

【機能】
プレビューを一時停止、または一時停止したプレビューを再開します。

【書式】
Delphi ［ *Boolean =*］*previewcontrolname*.**Pause**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Start** メソッドで開始したプレビューを一時停止します。また、一時停止した状態で再度このメソッドを実行するとプレビューを再開します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2 や 11 が設定されます。

SetStreamFormat(プレビューコントロール／カスタムメソッド)

【機能】
フォーマット機能(出力サイズなど)を設定します。

【書式】
Delphi [*Boolean* =]*previewcontrolname*.**SetStreamFormat**(Index: Integer)

【引数】

名称	内容
Index	インデックス(0 から **GetStreamFormats** メソッドの戻り値-1 の範囲内)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
プレビュー画像の出力サイズを変更する場合などに実行します。**ShowCapturePinDialog** メソッドを実行しても出力サイズの変更は可能です。プレビュー中に出力サイズなどを変更しても設定が反映されませんので、一旦プレビューを停止(**Stop** メソッドを実行)してから出力サイズなどを変更してください。
引数の Index に 0 から **GetStreamFormats** メソッドの戻り値-1 までの値を渡した場合は、そのインデックスに見合ったフォーマット機能を設定します。
戻り値が True の場合、**BitCount**,**Compression**,**VideoSize**,**VideoSubType** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2,5,11 のいずれかが設定されます。
SetStreamFormat メソッドを実行するには、事前に **Open** と **GetStreamFormats** メソッドを実行しておかなければなりません。

ShowCapturePinDialog(プレビューコントロール／カスタムメソッド)

【機能】
キャプチャーピンダイアログを表示します。

【書式】
Delphi ［ *Boolean =*］*previewcontrolname*.**ShowCapturePinDialog**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドを実行する場合は、事前に **Open** メソッドを実行して Web(USB)カメラと接続している必要があります。
戻り値が True の場合、**BitCount**,**Compression**,**FrameRate**,**VideoSize**,**VideoSubType** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2,11,202 のいずれかが設定されます。
プレビュー中に出力サイズなどを変更しても設定が反映されませんので、一旦プレビューを停止（**Stop** メソッドを実行）してから出力サイズなどを変更してください。

ShowFilterDialog(プレビューコントロール／カスタムメソッド)

【機能】
フィルタダイアログを表示します。

【書式】
Delphi ［ *Boolean* =］*previewcontrolname*.**ShowFilterDialog**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドを実行する場合は、事前に **Open** メソッドを実行して Web(USB)カメラと接続している必要があります。
戻り値が False の場合、**ErrorStatus** プロパティには 2,11,202 のいずれかが設定されます。

Start(プレビューコントロール／カスタムメソッド)

【機能】
プレビューを開始します。

【書式】
Delphi ［ *Boolean =*］*previewcontrolname*.**Start**(Mode: TVIkVideoDisplayMode)

【TVIkVideoDisplayMode 型】
ユニット
IkWebCamera
type
TVIkVideoDisplayMode = (vikVideoScale = 0, vikVideoStretch = 1);

【引数】

名称	内容
Mode	表示モード vikVideoScale:スケール(映像とコントロールの縦横比を保持して表示) vikVideoStretch:ストレッチ(映像をコントロールの縦横サイズに合わせて表示)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数の Mode には映像をどのように表示するかを渡します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2 や 11 が設定されます。

Stop(プレビューコントロール／カスタムメソッド)

【機能】
プレビューを停止します。

【書式】
Delphi ［ *Boolean =*］*previewcontrolname*.**Stop**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Start** メソッドで開始したプレビューを停止します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 11 が設定されます。

TakeSnapshot(プレビューコントロール／カスタムメソッド)

【機能】
静止画を取得します。

【書式】
Delphi ［ *Boolean =*］*previewcontrolname*.**TakeSnapshot**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Start** メソッドで開始したプレビューの静止画を取得します。静止画は **Snapshot** イベントで取得できます。
戻り値が False の場合、**ErrorStatus** プロパティには 2,4,11 のいずれかが設定されます。4 の場合、プレビューから静止画を取得することはできません。**ShowCapturePinDialog** メソッドを実行して色空間/圧縮方法を確認してください。

Snapshot（プレビューコントロール／カスタムイベント）

【機能】

TakeSnapshot メソッドが成功すると発生します。

【書式】

Delphi　*previewcontrolname*.Snapshot(Sender: TObject; ImageHandle: THandle; var FreeMemory: Boolean)

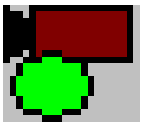
【引数】

名称	内容
ImageHandle	静止画のビットマップを示すメモリハンドル
FreeMemory	ImageHandle をイベント終了後に解放するかどうかを設定。既定値は True。 解放する場合は True、解放しない場合は False を設定します。

【解説】

引数の FreeMemory が True の場合は、取得した静止画のビットマップはイベント終了後に解放されますので、ImageHandle を残しておきたい場合は、イメージキットコントロールの CopyImage メソッドを実行するかファイルなどに保存してください。

1－5. レコードコントロール



レコードコントロール
Web(USB)カメラと接続して、映像を表示しながら、指定したファイルに映像を保存します。
保存の開始、停止、一時停止ができます。
※Delphi 専用のコントロールです（C++Builder へは非対応）。

継承
TCustomControl
ユニット
IkRecord

●プロパティ一覧（アルファベット順）

カスタムプロパティ	内容
Appearance	レコードコントロールの縁の設定
BitCount	キャプチャー画像の 1 ピクセルあたりのビット数
BitRate	ビットレートを設定
BorderColor	レコードコントロールの縁の色
BorderVisible	レコードコントロールの縁の描画設定
Color	レコードコントロールの背景色
Compression	キャプチャー画像の圧縮フォーマット
Cursor	レコードコントロールの描画領域内におけるマウスカーソルの形状
Enabled	マウスやキーボードイベントへの応答可否
ErrorStatus	エラーの種類を示す
Format	ファイルの保存形式
FrameRate	フレームレートを取得
Handle	レコードコントロールのウィンドウハンドル
MaximumBitRate	最大のビットレート
MaximumFrameRate	最大のフレームレート
MinimumBitRate	最小のビットレート
MinimumFrameRate	最小のフレームレート
State	現在の状態を取得
VideoDevices	ビデオデバイスの名称
VideoSize	キャプチャー画像の幅と高さを取得
VideoSubType	メディアサンプルのサブタイプを指定する GUID

【ImageKit9/10 ActiveX との違い】
削除されたプロパティ： MouseCursorFile
※MouseCursorFile プロパティは Cursor プロパティで代用

変更されたプロパティ:
BackColor --> Color
HWND --> Handle
MouseCursorType --> Cursor
VideoHeight,VideoWidth --> VideoSize

●メソッド一覧（アルファベット順）

カスタムメソッド	内容
Close	Web(USB)カメラとの接続を閉じる
GetStreamFormat	指定したフォーマット機能を取得
GetStreamFormats	フォーマット機能セットを取得
GetVideoDevices	ビデオデバイスの名称を取得
Open	キャプチャーモードで Web(USB)カメラと接続
Pause	キャプチャーの一時停止、または一時停止したキャプチャーの再開
SetStreamFormat	指定したフォーマット機能を設定

ShowCapturePinDialog	キャプチャーピンダイアログの表示
ShowFilterDialog	フィルタダイアログの表示
Start	キャプチャーの開始
Stop	キャプチャーの停止

Appearance(レコードコントロール／カスタムプロパティ)

【機能】
レコードコントロールの縁の表示を設定します。

【書式】
Delphi *recordcontrolname*.**Appearance** [= *TVIkAppearance*]

【TVIkAppearance 型】
ユニット
IkInit
type
TVIkAppearance = (vikFlat, vikLowered, vikRaised);

【設定値】

値	説明
vikFlat	Flat
vikLowered	凹
vikRaised	凸

【解説】
BorderVisible プロパティが True の場合に有効です。
vikFlat の場合、**BorderColor** プロパティに設定されている色で縁を描画します。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
列挙型の識別子の先頭に v が付加されました(ActiveX は ikFlat, ikLowered, ikRaised)。

BorderColor,Color(レコードコントロール／カスタムプロパティ)

【機能】
色情報を設定します。

【書式】
※BorderColor にて説明(Color も同様な使い方)
Delphi　*recordcontrolname*.BorderColor [= *TColor*]

【設定値】
BorderColor はレコードコントロールの縁の色。
Color はレコードコントロールの背景色。

【解説】
値を設定する場合は、色定数(clRed など)や RGB(Red,Green,Blue)の戻り値などを与えます。
BorderColor プロパティは Appearance プロパティが vikFlat で BorderVisible プロパティが True の場合に有効です。
TColor 型については Delphi のヘルプを参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
Color プロパティの名称が BackColor から変更されました。

BitCount (レコードコントロール／カスタムプロパティ)

【機能】
キャプチャー画像の 1 ピクセルあたりのビット数を示します。

【書式】
Delphi　*recordcontrolname*.BitCount [= *Word*]

【参照値】
1 ピクセルあたりのビット数。

【解説】
GetStreamFormat,GetStreamFormats,Open,SetStreamFormat,ShowCapturePinDialog メソッドを実行することにより、値が更新されます。

【値の設定】 不可

【値の参照】 実行時

BitRate(レコードコントロール／カスタムプロパティ)

【機能】
ビットレートを取得または設定します。

【書式】
Delphi *recordcontrolname*.**BitRate** [= *DWord*]

【設定値】
ビットレート(ビット/秒)。

【解説】
圧縮形式で保存する場合、ビットレートを大きくすると高画質となり、小さくすると粗くなります。
設定値は **Format** プロパティが vikAsf の場合に有効です。
Open,**ShowCapturePinDialog** メソッドを実行することにより、値が更新されます。
また、**SetStreamFormat** メソッドを実行する前に値を設定すると、その値が有効になります。

【値の設定】 実行時

【値の参照】 実行時

BorderVisible(レコードコントロール／カスタムプロパティ)

【機能】
レコードコントロールの縁を描画するかどうかを設定します。

【書式】
Delphi *recordcontrolname*.**BorderVisible** [= *Boolean*]

【設定値】

値	説明
True	縁を描画する
False	縁を描画しない

【値の設定】 デザイン時、実行時

【値の参照】 実行時

Compression(レコードコントロール／カスタムプロパティ)

【機能】
キャプチャー画像の圧縮フォーマットを示します。

【書式】
Delphi *recordcontrolname*.Compression [= *DWord*]

【参照値】
圧縮フォーマット。

【解説】
GetStreamFormat,GetStreamFormats,Open,SetStreamFormat,ShowCapturePinDialog メソッドを実行することにより、値が更新されます。
画像が非圧縮の場合は 0(BI_RGB)や 3(BI_BITFIELDS)が設定されます。()内の説明は Windows API で使用する定数と同じ意味です。
画像が圧縮されている場合は、FOURCC コードが割り当てられます。

【値の設定】 不可

【値の参照】 実行時

Cursor(レコードコントロール／カスタムプロパティ)

【機能】
レコードコントロールの描画領域内におけるマウスカーソルの形状を取得または設定します。

【書式】
Delphi　*recordcontrolname*.**Cursor** [= *TCursor*]

【設定値】
crDefault や crArrow など。詳しくは Delphi のヘルプを参照のこと。

【解説】
デフォルト値は crDefault です。
カスタムカーソルを割り当てる場合は、Delphi のヘルプの TControl.Cursor の例を参照してください。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
MouseCursorFile,MouseCursorType プロパティの機能が **Cursor** プロパティに統一されました。

Enabled(レコードコントロール／カスタムプロパティ)

【機能】
レコードコントロールのマウスやキーボードイベントに応答するかどうかを設定します。

【書式】
Delphi　*recordcontrolname*.**Enabled** [= *Boolean*]

【設定値】

値	説明
True	マウスやキーボードイベントを有効
False	マウスやキーボードイベントを無効

【解説】
True を設定するとマウスダウンやキーダウンなどのイベントに応答できますが、False の場合は該当イベントが発生しません。

【値の設定】 デザイン時、実行時

【値の参照】 実行時

ErrorStatus(レコードコントロール／カスタムプロパティ)

【機能】
レコードコントロールを使用して起きたエラーの種類を示します。

【書式】
Delphi *recordcontrolname*.**ErrorStatus** [= *Smallint*]

【参照値】

値	説明
0	正常終了(エラーなし)
2	その他のエラー
4	サポート外機能
5	設定値が不正
11	実行順序エラー
201	DirectX の環境が整っていない
202	デバイスが接続されていない
203	COM インターフェイスに関するエラー

【解説】
レコードコントロールのメソッドを実行することにより、値を参照できます。

【値の設定】 不可

【値の参照】 実行時

Format(レコードコントロール／カスタムプロパティ)

【機能】
ファイルの保存形式を取得または設定します。

【書式】
Delphi *recordcontrolname*.**Format** [= *TVIkVideoFormat*]

【TVIkVideoFormat 型】
ユニット
IkRecord
type
TVIkVideoFormat = (vikAvi, vikAsf);

【設定値】

値	説明
vikAvi	AVI 形式で保存(非圧縮)
vikAsf	WMV 形式で保存(圧縮)

【解説】
Open メソッドを実行する前に値を設定してください。デフォルトは vikAvi です。
WMV 形式で保存する場合は **BitRate** プロパティにも適切な値を設定してください。

【値の設定】 実行時

【値の参照】 実行時

FrameRate(レコードコントロール／カスタムプロパティ)

【機能】
フレームレートを取得または設定します。

【書式】
Delphi *recordcontrolname*.**FrameRate** [= *Double*]

【設定値】
フレームレート(フレーム/秒)。

【解説】
GetStreamFormat,**GetStreamFormats**,**Open**,**ShowCapturePinDialog** メソッドを実行することにより、値が更新されます。
また、**SetStreamFormat** メソッドを実行する前に値を設定すると、その値が有効になります。

【値の設定】 実行時

【値の参照】 実行時

Handle(レコードコントロール／カスタムプロパティ)

【機能】
レコードコントロールのウィンドウハンドルを示します。

【書式】
Delphi *recordcontrolname*.**Handle** [= *HWND*]

【参照値】
レコードコントロールのウィンドウハンドル。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
プロパティの名称が **HWND** から変更されました。

MaximumBitRate,MinimumBitRate(レコードコントロール／カスタムプロパティ)

【機能】
最大/最小のビットレートを取得します。

【書式】
※MaximumBitRate にて説明(MinimumBitRate も同様な使い方)
Delphi *recordcontrolname*.MaximumBitRate [= *DWord*]

【参照値】
ビットレート(ビット/秒)。

【解説】
GetStreamFormat メソッドを実行することにより、値が更新されます。
BitRate プロパティに設定する値の目安となりますが、ビデオデバイスによっては最大/最小の範囲外の値を設定することも可能です。

【値の設定】 実行時

【値の参照】 実行時

MaximumFrameRate,MinimumFrameRate(レコードコントロール／カスタムプロパティ)

【機能】
最大/最小のフレームレートを取得します。

【書式】
※MaximumFrameRate にて説明(MinimumFrameRate も同様な使い方)
Delphi *recordcontrolname*.MaximumFrameRate [= *Double*]

【参照値】
フレームレート(フレーム/秒)。

【解説】
GetStreamFormat メソッドを実行することにより、値が更新されます。
FrameRate プロパティに値を設定する場合の目安となります。

【値の設定】 実行時

【値の参照】 実行時

State(レコードコントロール／カスタムプロパティ)

【機能】
現在の状態を取得します。

【書式】
Delphi *recordcontrolname*.**State** [= *TVIkVideoStates*]

【TVIkVideoStates 型】
ユニット
IkWebCamera
type
TVIkVideoState = (vikClosed = 0, vikOpen = 1, vikRunning = 2, vikPaused = 4, vikStopped = 8);
TVIkVideoStates = set of TVIkVideoState;

【参照値】

値	説明
vikClosed	接続を閉じた、もしくは接続されていない
vikOpen	接続しています
vikRunning	実行中
vikPaused	一時停止しました
vikStopped	停止しました

【解説】
Close,**Open**,**Pause**,**Start**,**Stop** メソッド実行後に更新されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
集合型に変更されました(ActiveX は ikClosed, ikOpen, ikRunning, ikPaused, ikStopped。複数の組み合わせの場合、論理和となる)。

VideoDevices(レコードコントロール／カスタムプロパティ)

【機能】
ビデオデバイスの名称を取得します。

【書式】
Delphi　*recordcontrolname*.**VideoDevices** [= *TStrings*]

【参照値】
ビデオデバイスの名称。

【解説】
GetVideoDevices メソッドが成功した場合にビデオデバイスの名称が設定されます。
接続されているビデオデバイスの数は **VideoDevices.Count** で取得できます。
TStrings については Delphi のヘルプを参照してください。

【値の設定】 不可

【値の参照】 実行時

VideoSize(レコードコントロール／カスタムプロパティ)

【機能】
キャプチャー画像の高さと幅を取得します。

【書式】
Delphi *recordcontrolname*.**VideoSize** [= *TSize*]

【参照値】
VideoSize.cx はキャプチャー画像の幅。
VideoSize.cy はキャプチャー画像の高さ。

【解説】
GetStreamFormat,**GetStreamFormats**,**Open**,**SetStreamFormat**,**ShowCapturePinDialog** メソッドを実行することにより、値が更新されます。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
VideoHeight,**VideoWidth** プロパティの機能が **VideoSize** プロパティに統一されました。

VideoSubType(レコードコントロール／カスタムプロパティ)

【機能】
メディアサンプルのサブタイプを指定する GUID を取得します。

【書式】
Delphi *recordcontrolname*.**VideoSubType** [= *TGUID*]

【参照値】
GUID。

【解説】
GetStreamFormat,**GetStreamFormats**,**Open**,**SetStreamFormat**,**ShowCapturePinDialog** メソッドを実行することにより、値が更新されます。
キャプチャー画像が圧縮されている場合、16 進表記で最初の 8 桁が **FOURCC** コードになります。

【値の設定】 不可

【値の参照】 実行時

【ImageKit9/10 ActiveX との違い】
文字列型からレコード型に変更されました。

Close（レコードコントロール／カスタムメソッド）

【機能】
Web(USB)カメラとの接続を閉じます。

【書式】
Delphi ［ *Boolean* =］*recordcontrolname*.**Close**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Open** メソッドで接続した Web(USB)カメラを閉じます。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 11 が設定されます。**Open** メソッドを実行せずに **Close** メソッドを実行した場合が該当します。

GetStreamFormat（レコードコントロール／カスタムメソッド）

【機能】
指定したインデックスのフォーマット機能を取得します。

【書式】
Delphi ［ *Boolean* =］*recordcontrolname*.**GetStreamFormat**(Index: Integer)

【引数】

名称	内容
Index	インデックス（-1 から **GetStreamFormats** メソッドの戻り値-1 の範囲内）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数の Index に-1 を渡した場合は現在選択されているフォーマット機能を取得します。Index に 0 から **GetStreamFormats** メソッドの戻り値-1 までの値を渡した場合は、そのインデックスに見合ったフォーマット機能を取得します。
戻り値が True の場合、**BitCount**,**Compression**,**FrameRate**,**MaximumBitRate**,**MaximumFrameRate**,**MinimumBitRate**, **MinimumFrameRate**,**VideoSize**,**VideoSubType** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 5 や 11 が設定されます。
GetStreamFormat メソッドを実行するには、事前に **Open** と **GetStreamFormats** メソッドを実行しておかなければなりません。

GetStreamFormats（レコードコントロール／カスタムメソッド）

【機能】
フォーマット機能セットを取得します。

【書式】
Delphi ［ *Integer* = ］*recordcontrolname*.GetStreamFormats

【引数】
ありません。

【戻り値】
取得した数を返します。

【解説】
戻り値が 1 以上の場合、GetStreamFormat メソッドを実行することにより、対応している出力サイズなどの情報を取得できます。
戻り値が 0 の場合、ErrorStatus プロパティには 2 や 11 が設定されます。
GetStreamFormats メソッドを実行するには、事前に Open メソッドを実行しておかなければなりません。

GetVideoDevices（レコードコントロール／カスタムメソッド）

【機能】
ビデオデバイスの名称を取得します。

【書式】
Delphi ［ *Integer* = ］*recordcontrolname*.**GetVideoDevices**

【引数】
ありません。

【戻り値】
取得した数を返します。

【解説】
戻り値が 1 以上の場合、**VideoDevices** プロパティにビデオデバイスの名称が設定されます。
戻り値が 0 の場合、**ErrorStatus** プロパティには 201 や 202 が設定されます。

Open(レコードコントロール／カスタムメソッド)

【機能】
キャプチャーモードで Web(USB)カメラと接続します。

【書式】
Delphi [*Boolean* =]*recordcontrolname*.**Open**(const FileName: string; Index: Integer)

【引数】

名称	内容
FileName	保存するファイルの名称
Index	Web(USB)カメラのインデックス番号

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数の FileName で指定されたファイルに AVI または WMV 形式(**Format** プロパティによる)で動画が保存されます。
引数の Index には **GetVideoDevices** メソッドで取得されるビデオデバイスの名称を示すインデックス(先頭のビデオデバイスを接続する場合は 0)を渡します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 5,11,201,202,203 のいずれかが設定されます。

注意事項
プレビュー(プレビューコントロール)とキャプチャーを同時に行うことはできません。キャプチャーを行う場合はプレビューコントロールの **Close** メソッドを実行して一旦 Web(USB)カメラとの接続を閉じ、再度レコードコントロールの **Open** メソッドを実行して接続してください。

Pause(レコードコントロール／カスタムメソッド)

【機能】
キャプチャーを一時停止、または一時停止したキャプチャーを再開します。

【書式】
Delphi ［ *Boolean* =］*recordcontrolname*.**Pause**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Start** メソッドで開始したキャプチャーを一時停止します。また、一時停止した状態で再度このメソッドを実行するとキャプチャーを再開します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2 や 11 が設定されます。

SetStreamFormat（レコードコントロール／カスタムメソッド）

【機能】
フォーマット機能（出力サイズなど）を設定します。

【書式】
Delphi ［ *Boolean* =］*recordcontrolname*.**SetStreamFormat**(Index: Integer)

【引数】

名称	内容
Index	インデックス（0 から **GetStreamFormats** メソッドの戻り値-1 の範囲内）

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
キャプチャー画像の出力サイズを変更する場合などに実行します。**ShowCapturePinDialog** メソッドを実行しても出力サイズの変更は可能です。キャプチャー中に出力サイズなどを変更しても設定が反映されませんので、一旦キャプチャーを停止（**Stop** メソッドを実行）してから出力サイズなどを変更してください。
引数の Index に 0 から **GetStreamFormats** メソッドの戻り値-1 までの値を渡した場合は、そのインデックスに見合ったフォーマット機能を設定します。
戻り値が True の場合、**BitCount**,**Compression**,**VideoSize**,**VideoSubType** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2,5,11 のいずれかが設定されます。
SetStreamFormat メソッドを実行するには、事前に **Open** と **GetStreamFormats** メソッドを実行しておかなければなりません。

ShowCapturePinDialog（レコードコントロール／カスタムメソッド）

【機能】
キャプチャーピンダイアログを表示します。

【書式】
Delphi ［ *Boolean* =］*recordcontrolname*.**ShowCapturePinDialog**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドを実行する場合は、事前に **Open** メソッドを実行して Web(USB)カメラと接続している必要があります。
戻り値が True の場合、**BitCount**,**BitRate**,**Compression**,**FrameRate**,**VideoSize**,**VideoSubType** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2,11,202 のいずれかが設定されます。
キャプチャー中に出力サイズなどを変更しても設定が反映されませんので、一旦キャプチャーを停止（**Stop** メソッドを実行）してから出力サイズなどを変更してください。

ShowFilterDialog(レコードコントロール／カスタムメソッド)

【機能】
フィルタダイアログを表示します。

【書式】
Delphi [*Boolean* =]*recordcontrolname*.**ShowFilterDialog**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドを実行する場合は、事前に **Open** メソッドを実行して Web(USB)カメラと接続している必要があります。
戻り値が False の場合、**ErrorStatus** プロパティには 2,11,202 のいずれかが設定されます。

Start(レコードコントロール／カスタムメソッド)

【機能】
キャプチャーを開始します。

【書式】
Delphi ［ *Boolean* =］*recordcontrolname*.**Start**(Mode: TVIkVideoDisplayMode)

【TVIkVideoDisplayMode 型】
ユニット
IkWebCamera
type
TVIkVideoDisplayMode = (vikVideoScale = 0, vikVideoStretch = 1);

【引数】

名称	内容
Mode	表示モード vikVideoScale:スケール(映像とコントロールの縦横比を保持して表示) vikVideoStretch:ストレッチ(映像をコントロールの縦横サイズに合わせて表示)

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
引数の Mode には映像をどのように表示するかを渡します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 2 や 11 が設定されます。

Stop(レコードコントロール／カスタムメソッド)

【機能】
キャプチャーを停止します。

【書式】
Delphi ［ *Boolean* =］*recordcontrolname*.**Stop**

【引数】
ありません。

【戻り値】
成功の場合は True、失敗の場合は False を返します。

【解説】
このメソッドは、**Start** メソッドで開始したキャプチャーを停止します。
戻り値が True の場合、**State** プロパティが更新されます。
戻り値が False の場合、**ErrorStatus** プロパティには 11 が設定されます。

2. 作成したアプリケーションの配布

ImageKit10 を使用して作成したアプリケーションをユーザに配布する場合は、そのアプリケーションの実行時に以下のファイルが必要になりますので、忘れずに配布リストに加えてください。

A.Windows アプリケーション

(1)設計時パッケージの場合

ImageKit10 の DLL ファイルが必要です。

C++Builder XE3(32ビット)/C++Builder XE4(32ビット)/C++Builder XE5(32ビット)/C++Builder XE6(32ビット)/C++Builder XE7(32 ビット)/C++Builder XE8(32 ビット)/C++Builder 10 Seattle(32 ビット)/C++Builder 10.1 Berlin(32 ビット)/C++Builder 10.2 Tokyo(32 ビット) /C++Builder 10.3 Rio(32 ビット):
Delphi XE3(32 ビット)/Delphi XE4(32 ビット)/Delphi XE5(32 ビット)/ Delphi XE6(32 ビット)/Delphi XE7(32 ビット)/Delphi XE8(32 ビット) /Delphi 10 Seattle(32 ビット)/Delphi 10.1 Berlin(32 ビット)/Delphi 10.2 Tokyo(32 ビット)/Delphi 10.3 Rio(32 ビット):

- Ik10Com.dll(必ず必要)
- Ik10Effect.dll(必ず必要)
- Ik10File.dll(必ず必要)
  - Ik10Bmp.dll(BMP 形式のロード、保存を行う場合に必要)
  - Ik10Dxf.dll(DXF 形式のロード、保存を行う場合に必要)
  - Ik10Emf.dll(EMF 形式のロード、保存を行う場合に必要)
  - Ik10Fpx.dll(FPX 形式のロード、保存を行う場合に必要)
  - Ik10Gif.dll(GIF 形式のロード、保存を行う場合に必要)
  - Ik10J2k.dll(JPEG2000 形式のロード、保存を行う場合に必要)
  - Ik10Jpeg.dll(JPEG 形式のロード、保存を行う場合に必要)
  - Ik10Pcx.dll(PCX 形式のロード、保存を行う場合に必要)
  - Ik10Pdf.dll(PDF 形式の保存を行う場合に必要)
  - Ik10Png.dll(PNG 形式のロード、保存を行う場合に必要)
  - Ik10Svg.dll(SVG 形式のロード、保存を行う場合に必要)
  - Ik10SxfP21.dll(SXF p21 形式のロード、保存を行う場合に必要)
  - Ik10SxfSfc.dll(SXF sfc 形式のロード、保存を行う場合に必要)
  - Ik10Tiff.dll(TIFF 形式のロード、保存を行う場合に必要)
  - Ik10TransFile.dll(FTP・HTTP 転送を行う場合に必要)
  - Ik10Wmf.dll(WMF 形式のロード、保存を行う場合に必要)
- Ik10Print.dll(必ず必要)
- Ik10RasToVect.dll(ラスタイベージからベクトルイメージへの変換を行う場合に必要)
- Ik10Scan.dll(必ず必要)
- Ik10VectCom.dll(ベクトルイメージを扱う場合に必要)

C++Builder XE3(64ビット)/C++Builder XE4(64ビット)/C++Builder XE5(64ビット)/C++Builder XE6(64ビット)/C++Builder XE7(64 ビット)/C++Builder XE8(64 ビット)/C++Builder 10 Seattle(64 ビット)/C++Builder 10.1 Berlin(64 ビット)/C++Builder 10.2 Tokyo(64 ビット)/C++Builder 10.3 Rio(64 ビット):
Delphi XE3(64 ビット)/Delphi XE4(64 ビット)/Delphi XE5(64 ビット)/Delphi XE6(64 ビット)/Delphi XE7(64 ビット)/Delphi XE8(64 ビット)/Delphi 10 Seattle(64 ビット)/Delphi 10.1 Berlin(64 ビット)/Delphi 10.2 Tokyo(64 ビット)/Delphi 10.3 Rio(64 ビット):

- Ik10Com64.dll(必ず必要)
- Ik10Effect64.dll(必ず必要)
- Ik10File64.dll(必ず必要)
  - Ik10Bmp64.dll(BMP 形式のロード、保存を行う場合に必要)
  - Ik10Dxf64.dll(DXF 形式のロード、保存を行う場合に必要)
  - Ik10Emf64.dll(EMF 形式のロード、保存を行う場合に必要)
  - Ik10Fpx64.dll(FPX 形式のロード、保存を行う場合に必要)
  - Ik10Gif64.dll(GIF 形式のロード、保存を行う場合に必要)
  - Ik10J2k64.dll(JPEG2000 形式のロード、保存を行う場合に必要)
  - Ik10Jpeg64.dll(JPEG 形式のロード、保存を行う場合に必要)
  - Ik10Pcx64.dll(PCX 形式のロード、保存を行う場合に必要)
  - Ik10Pdf64.dll(PDF 形式の保存を行う場合に必要)

Ik10Png64.dll(PNG 形式のロード、保存を行う場合に必要)
Ik10Svg64.dll(SVG 形式のロード、保存を行う場合に必要)
Ik10SxfP2164.dll(SXF p21 形式のロード、保存を行う場合に必要)
Ik10SxfSfc64.dll(SXF sfc 形式のロード、保存を行う場合に必要)
Ik10Tiff64.dll(TIFF 形式のロード、保存を行う場合に必要)
Ik10TransFile64.dll(FTP・HTTP 転送を行う場合に必要)
Ik10Wmf64.dll(WMF 形式のロード、保存を行う場合に必要)
Ik10Print64.dll(必ず必要)
Ik10RasToVect64.dll(ラスタイベージからベクトルイメージへの変換を行う場合に必要)
Ik10Scan64.dll(必ず必要)
Ik10VectCom64.dll(ベクトルイメージを扱う場合に必要)

(2)実行時パッケージの場合
ImageKit10 の BPL ファイルと DLL ファイルが必要です。

C++Builder XE3(32ビット)/C++Builder XE4(32ビット)/C++Builder XE5(32ビット)/C++Builder XE6(32ビット)/C++Builder XE7(32 ビット)/C++Builder XE8(32 ビット)/C++Builder 10 Seattle(32 ビット)/C++Builder 10.1 Berlin(32 ビット)/C++Builder 10.2 Tokyo(32 ビット)/C++Builder 10.3 Rio(32 ビット):
Delphi XE3(32 ビット)/Delphi XE4(32 ビット)/Delphi XE5(32 ビット)/ Delphi XE6(32 ビット)/Delphi XE7(32 ビット)/Delphi XE8(32 ビット)/Delphi 10 Seattle(32 ビット)/Delphi 10.1 Berlin(32 ビット)/Delphi 10.2 Tokyo(32 ビット)/Delphi 10.3 Rio(64 ビット):

BPL ファイル
必ず必要なファイル

C++Builder XE3:	ImgKit10CoreD17.bpl
C++Builder XE4:	ImgKit10CoreD18.bpl
C++Builder XE5:	ImgKit10CoreD19.bpl
C++Builder XE6:	ImgKit10CoreD20.bpl
C++Builder XE7:	ImgKit10CoreD21.bpl
C++Builder XE8:	ImgKit10CoreD22.bpl
C++Builder 10 Seattle:	ImgKit10CoreD23.bpl
C++Builder 10.1 Berlin:	ImgKit10CoreD24.bpl
C++Builder 10.2 Tokyo:	ImgKit10CoreD25.bpl
C++Builder 10.3 Rio:	ImgKit10CoreD26.bpl
Delphi XE3:	ImgKit10CoreD17.bpl
Delphi XE4:	ImgKit10CoreD18.bpl
Delphi XE5:	ImgKit10CoreD19.bpl
Delphi XE6:	ImgKit10CoreD20.bpl
Delphi XE7:	ImgKit10CoreD21.bpl
Delphi XE8:	ImgKit10CoreD22.bpl
Delphi 10 Seattle:	ImgKit10CoreD23.bpl
Delphi 10.1 Berlin:	ImgKit10CoreD24.bpl
Delphi 10.2 Tokyo:	ImgKit10CoreD25.bpl
Delphi 10.3 Rio:	ImgKit10CoreD26.bpl

イメージキットコントロールを使用する場合に必要なファイル

C++Builder XE3:	ImgKit10D17.bpl
C++Builder XE4:	ImgKit10D18.bpl
C++Builder XE5:	ImgKit10D19.bpl
C++Builder XE6:	ImgKit10D20.bpl
C++Builder XE7:	ImgKit10D21.bpl
C++Builder XE8:	ImgKit10D22.bpl
C++Builder 10 Seattle:	ImgKit10D23.bpl
C++Builder 10.1 Berlin:	ImgKit10D24.bpl
C++Builder 10.2 Tokyo:	ImgKit10D25.bpl
C++Builder 10.3 Rio:	ImgKit10D26.bpl
Delphi XE3:	ImgKit10D17.bpl
Delphi XE4:	ImgKit10D18.bpl
Delphi XE5:	ImgKit10D19.bpl
Delphi XE6:	ImgKit10D20.bpl

Delphi XE7:	ImgKit10D21.bpl
Delphi XE8:	ImgKit10D22.bpl
Delphi 10 Seattle:	ImgKit10D23.bpl
Delphi 10.1 Berlin:	ImgKit10D24.bpl
Delphi 10.2 Tokyo:	ImgKit10D25.bpl
Delphi 10.3 Rio:	ImgKit10D26.bpl

サムネイルコントロールを使用する場合に必要なファイル

C++Builder XE3:	ImgKit10ThumbD17.bpl
C++Builder XE4:	ImgKit10ThumbD18.bpl
C++Builder XE5:	ImgKit10ThumbD19.bpl
C++Builder XE6:	ImgKit10ThumbD20.bpl
C++Builder XE7:	ImgKit10ThumbD21.bpl
C++Builder XE8:	ImgKit10ThumbD22.bpl
C++Builder 10 Seattle:	ImgKit10ThumbD23.bpl
C++Builder 10.1 Berlin:	ImgKit10ThumbD24.bpl
C++Builder 10.2 Tokyo:	ImgKit10ThumbD25.bpl
C++Builder 10.3 Rio:	ImgKit10ThumbD26.bpl
Delphi XE3:	ImgKit10ThumbD17.bpl
Delphi XE4:	ImgKit10ThumbD18.bpl
Delphi XE5:	ImgKit10ThumbD19.bpl
Delphi XE6:	ImgKit10ThumbD20.bpl
Delphi XE7:	ImgKit10ThumbD21.bpl
Delphi XE8:	ImgKit10ThumbD22.bpl
Delphi 10 Seattle:	ImgKit10ThumbD23.bpl
Delphi 10.1 Berlin:	ImgKit10ThumbD24.bpl
Delphi 10.2 Tokyo:	ImgKit10ThumbD25.bpl
Delphi 10.3 Rio:	ImgKit10ThumbD26.bpl

プレイコントロール/プレビューコントロール/レコードコントロールを使用する場合に必要なファイル

Delphi XE3:	ImgKit10WebCamD17.bpl
Delphi XE4:	ImgKit10WebCamD18.bpl
Delphi XE5:	ImgKit10WebCamD19.bpl
Delphi XE6:	ImgKit10WebCamD20.bpl
Delphi XE7:	ImgKit10WebCamD21.bpl
Delphi XE8:	ImgKit10WebCamD22.bpl
Delphi 10 Seattle:	ImgKit10WebCamD23.bpl
Delphi 10.1 Berlin:	ImgKit10WebCamD24.bpl
Delphi 10.2 Tokyo:	ImgKit10WebCamD25.bpl
Delphi 10.3 Rio:	ImgKit10WebCamD26.bpl

DLL ファイル

Ik10Com.dll（必ず必要）
Ik10Effect.dll（必ず必要）
Ik10File.dll（必ず必要）
- Ik10Bmp.dll（BMP 形式のロード、保存を行う場合に必要）
- Ik10Dxf.dll（DXF 形式のロード、保存を行う場合に必要）
- Ik10Emf.dll（EMF 形式のロード、保存を行う場合に必要）
- Ik10Fpx.dll（FPX 形式のロード、保存を行う場合に必要）
- Ik10Gif.dll（GIF 形式のロード、保存を行う場合に必要）
- Ik10J2k.dll（JPEG2000 形式のロード、保存を行う場合に必要）
- Ik10Jpeg.dll（JPEG 形式のロード、保存を行う場合に必要）
- Ik10Pcx.dll（PCX 形式のロード、保存を行う場合に必要）
- Ik10Pdf.dll（PDF 形式の保存を行う場合に必要）
- Ik10Png.dll（PNG 形式のロード、保存を行う場合に必要）
- Ik10Svg.dll（SVG 形式のロード、保存を行う場合に必要）
- Ik10SxfP21.dll（SXF p21 形式のロード、保存を行う場合に必要）
- Ik10SxfSfc.dll（SXF sfc 形式のロード、保存を行う場合に必要）
- Ik10Tiff.dll（TIFF 形式のロード、保存を行う場合に必要）

Ik10TransFile.dll(FTP・HTTP 転送を行う場合に必要)
Ik10Wmf.dll(WMF 形式のロード、保存を行う場合に必要)
Ik10Print.dll(必ず必要)
Ik10RasToVect.dll(ラスタイベージからベクトルイメージへの変換を行う場合に必要)
Ik10Scan.dll(必ず必要)
Ik10VectCom.dll(ベクトルイメージを扱う場合に必要)

C++Builder XE3(64ビット)/C++Builder XE4(64ビット)/C++Builder XE5(64ビット)/C++Builder XE6(64ビット)/C++Builder XE7(64 ビット)/C++Builder XE8(64 ビット)/C++Builder 10 Seattle(64 ビット)/C++Builder 10.1 Berlin(64 ビット)/C++Builder 10.2 Tokyo(64 ビット)/C++Builder 10.3 Rio(64 ビット):
Delphi XE3(64 ビット)/Delphi XE4(64 ビット)/Delphi XE5(64 ビット)/Delphi XE6(64 ビット)/Delphi XE7(64 ビット)/Delphi XE8(64 ビット)/Delphi 10 Seattle(64 ビット)/Delphi 10.1 Berlin(64 ビット)/Delphi 10.2 Tokyo(64 ビット)/Delphi 10.3 Rio(64 ビット):

BPL ファイル

必ず必要なファイル

C++Builder XE3:	ImgKit10CoreD17.bpl
C++Builder XE4:	ImgKit10CoreD18.bpl
C++Builder XE5:	ImgKit10CoreD19.bpl
C++Builder XE6:	ImgKit10CoreD20.bpl
C++Builder XE7:	ImgKit10CoreD21.bpl
C++Builder XE8:	ImgKit10CoreD22.bpl
C++Builder 10 Seattle:	ImgKit10CoreD23.bpl
C++Builder 10.1 Berlin:	ImgKit10CoreD24.bpl
C++Builder 10.2 Tokyo:	ImgKit10CoreD25.bpl
C++Builder 10.3 Rio:	ImgKit10CoreD26.bpl
Delphi XE3:	ImgKit10CoreD17.bpl
Delphi XE4:	ImgKit10CoreD18.bpl
Delphi XE5:	ImgKit10CoreD19.bpl
Delphi XE6:	ImgKit10CoreD20.bpl
Delphi XE7:	ImgKit10CoreD21.bpl
Delphi XE8:	ImgKit10CoreD22.bpl
Delphi 10 Seattle:	ImgKit10CoreD23.bpl
Delphi 10.1 Berlin:	ImgKit10CoreD24.bpl
Delphi 10.2 Tokyo:	ImgKit10CoreD25.bpl
Delphi 10.3 Rio:	ImgKit10CoreD26.bpl

イメージキットコントロールを使用する場合に必要なファイル

C++Builder XE3:	ImgKit10D17.bpl
C++Builder XE4:	ImgKit10D18.bpl
C++Builder XE5:	ImgKit10D19.bpl
C++Builder XE6:	ImgKit10D20.bpl
C++Builder XE7:	ImgKit10D21.bpl
C++Builder XE8:	ImgKit10D22.bpl
C++Builder 10 Seattle:	ImgKit10D23.bpl
C++Builder 10.1 Berlin:	ImgKit10D24.bpl
C++Builder 10.2 Tokyo:	ImgKit10D25.bpl
C++Builder 10.3 Rio:	ImgKit10D26.bpl
Delphi XE3:	ImgKit10D17.bpl
Delphi XE4:	ImgKit10D18.bpl
Delphi XE5:	ImgKit10D19.bpl
Delphi XE6:	ImgKit10D20.bpl
Delphi XE7:	ImgKit10D21.bpl
Delphi XE8:	ImgKit10D22.bpl
Delphi 10 Seattle:	ImgKit10D23.bpl
Delphi 10.1 Berlin:	ImgKit10D24.bpl
Delphi 10.2 Tokyo:	ImgKit10D25.bpl
Delphi 10.3 Rio:	ImgKit10D26.bpl

サムネイルコントロールを使用する場合に必要なファイル

C++Builder XE3: ImgKit10ThumbD17.bpl
C++Builder XE4: ImgKit10ThumbD18.bpl
C++Builder XE5: ImgKit10ThumbD19.bpl
C++Builder XE6: ImgKit10ThumbD20.bpl
C++Builder XE7: ImgKit10ThumbD21.bpl
C++Builder XE8: ImgKit10ThumbD22.bpl
C++Builder 10 Seattle: ImgKit10ThumbD23.bpl
C++Builder 10.1 Berlin: ImgKit10ThumbD24.bpl
C++Builder 10.2 Tokyo: ImgKit10ThumbD25.bpl
C++Builder 10.3 Rio: ImgKit10ThumbD26.bpl
Delphi XE3: ImgKit10ThumbD17.bpl
Delphi XE4: ImgKit10ThumbD18.bpl
Delphi XE5: ImgKit10ThumbD19.bpl
Delphi XE6: ImgKit10ThumbD20.bpl
Delphi XE7: ImgKit10ThumbD21.bpl
Delphi XE8: ImgKit10ThumbD22.bpl
Delphi 10 Seattle: ImgKit10ThumbD23.bpl
Delphi 10.1 Berlin: ImgKit10ThumbD24.bpl
Delphi 10.2 Tokyo: ImgKit10ThumbD25.bpl
Delphi 10.3 Rio: ImgKit10ThumbD26.bpl

プレイコントロール/プレビューコントロール/レコードコントロールを使用する場合に必要なファイル
Delphi XE3: ImgKit10WebCamD17.bpl
Delphi XE4: ImgKit10WebCamD18.bpl
Delphi XE5: ImgKit10WebCamD19.bpl
Delphi XE6: ImgKit10WebCamD20.bpl
Delphi XE7: ImgKit10WebCamD21.bpl
Delphi XE8: ImgKit10WebCamD22.bpl
Delphi 10 Seattle: ImgKit10WebCamD23.bpl
Delphi 10.1 Berlin: ImgKit10WebCamD24.bpl
Delphi 10.2 Tokyo: ImgKit10WebCamD25.bpl
Delphi 10.3 Rio: ImgKit10WebCamD26.bpl

32 ビット用の bpl と 64 ビット用の bpl のファイル名は同じですのでお間違いのないようにお願いいたします。

DLL ファイル
Ik10Com64.dll(必ず必要)
Ik10Effect64.dll(必ず必要)
Ik10File64.dll(必ず必要)
Ik10Bmp64.dll(BMP 形式のロード、保存を行う場合に必要)
Ik10Dxf64.dll(DXF 形式のロード、保存を行う場合に必要)
Ik10Emf64.dll(EMF 形式のロード、保存を行う場合に必要)
Ik10Fpx64.dll(FPX 形式のロード、保存を行う場合に必要)
Ik10Gif64.dll(GIF 形式のロード、保存を行う場合に必要)
Ik10J2k64.dll(JPEG2000 形式のロード、保存を行う場合に必要)
Ik10Jpeg64.dll(JPEG 形式のロード、保存を行う場合に必要)
Ik10Pcx64.dll(PCX 形式のロード、保存を行う場合に必要)
Ik10Pdf64.dll(PDF 形式の保存を行う場合に必要)
Ik10Png64.dll(PNG 形式のロード、保存を行う場合に必要)
Ik10Svg64.dll(SVG 形式のロード、保存を行う場合に必要)
Ik10SxfP2164.dll(SXF p21 形式のロード、保存を行う場合に必要)
Ik10SxfSfc64.dll(SXF sfc 形式のロード、保存を行う場合に必要)
Ik10Tiff64.dll(TIFF 形式のロード、保存を行う場合に必要)
Ik10TransFile64.dll(FTP・HTTP 転送を行う場合に必要)
Ik10Wmf64.dll(WMF 形式のロード、保存を行う場合に必要)
Ik10Print64.dll(必ず必要)
Ik10RasToVect64.dll(ラスタイベージからベクトルイメージへの変換を行う場合に必要)
Ik10Scan64.dll(必ず必要)
Ik10VectCom64.dll(ベクトルイメージを扱う場合に必要)

BPL や DLL ファイルは、Windows¥System32（64 ビット OS で 32 ビットモジュールを使用する場合は Windows¥SysWOW64）もしくはアプリケーションと同じフォルダにコピーしてお使いください。その他、ImageKit10 以外のファイルでセットアップディスクを作成時に必要なファイルについては、その開発環境のマニュアルなどをご覧ください。
また、ImageKit10（ActiveX もしくは VCL）で作成された別アプリケーションがインストールされたことによる影響などを避けたい場合は、アプリケーションと同じフォルダに BPL や DLL をコピーすることをお勧めいたします。

B.Web アプリケーション

サーバー側で BPL もしくは DLL を実行する場合

A.Windows アプリケーションを参照

A と B 共通

配布する環境に「Microsoft Visual C++ 2019 再頒布可能パッケージ」をインストールしてください。「Microsoft Visual C++ 2019 再頒布可能パッケージ」には (x86),(x64) の 2 種類ありますので、配布するアプリケーションに応じて選択してください。
※「Microsoft Visual C++ 2019 再頒布可能パッケージ」は Microsoft 社のオフィシャルサイトからダウンロードすることができます。
サーバで運用する場合は、サーバ毎に Server ランタイムライセンスが必要です。

（注意）

製品に付属のドキュメントや設計時用のパッケージファイルを再配布することはできません。

以前の ImageKit との互換性

・「ActiveX（OCX）コントロールとして使用していた場合」
以前の ImageKit を使用していたプログラムで、デザインやコードに関しては、ImageKit10 VCL の新しいコントロールに自動的に置き換えることはできません。相違点を考慮した上で手動で作業することになります。
ただし、以前の ImageKit との共存は可能です。例えば、同じフォーム上に新たに ImageKit10 VCL のコントロールを配置して使用することは問題ありません。また、以前の ImageKit のメモリハンドルを ImageKit10 VCL のコントロールへ代入して新機能を利用することも可能です。

・「VCL コントロールとして使用していた場合」
ImageKit10 VCL を開発環境に組み込むと ImageKit6 VCL のサムネイルコントロールや ImageKit7/8/9 VCL を使用していたプログラムをそのまま利用することができます。ただし、いくつか変更点がありますので、その点を考慮してください。
C++Builder で実行時パッケージとして利用していた場合、ImageKit6/7/8/9 と 10 ではファイル名が異なりますので、プロジェクトファイルの該当部分を変更する必要があります。

以前の ImageKit で提供していたコモンコントロール、ディスプレイコントロール、エフェクトコントロール、ファイルコントロール、プリントコントロール、スキャンコントロール、スライドショウコントロールの 7 つは ImageKit8 VCL からは提供しておりません。

ImageKit5
(1)プロパティ名の変更（ImageKit5 → ImageKit10 VCL）
サムネイルコントロール
BackColor --> Color
Hwnd --> Handle
(2)その他
メソッドやイベントの引数および戻り値を変更。詳しくはリファレンスの【ImageKit5 ActiveX との違い】を参照してください。

プロパティやメソッドとイベント引数ついて
文字列型
ActiveX の WideString 型や BSTR 型は、VCL では UnicodeString 型に変更されました。
整数型
ActiveX を C++Builder で使用する際の long 型が int 型に変更されました。
論理型
ActiveX を Delphi で使用する際の WordBool 型が Boolean 型に変更されました。

ImageKit6
(1)プロパティの削除（ImageKit6 → ImageKit10 VCL）
サムネイルコントロール
EnableOLEDrag, MouseCursorTypeFile
※MouseCursorTypeFile プロパティは Cursor プロパティにて代用
(2)プロパティ名の変更（ImageKit6 → ImageKit10 VCL）
サムネイルコントロール
BackColor --> Color
ForeColorUp --> ForeColor
Hdc --> Canvas.Handle
Hwnd --> Handle
MouseCursor --> SelectCursor
MouseCursorType --> Cursor
PictureUp --> Picture
PictureUpFile --> PictureFile
(3)仕様変更
サムネイルコントロールの **ExifAutoRotate** プロパティを追加したことにより、表示される画像の向きが異なる場合があります。
以前の ImageKit と同じ動作とする場合は **ExifAutoRotate** プロパティを False に設定してください。
サムネイルコントロールの **MouseMoveOnThumb** イベントの発生条件を変更。
(4)その他
詳しくはリファレンスの【ImageKit6 ActiveX との違い】を参照してください。

プロパティやメソッドとイベント引数ついて

文字列型
ActiveX の WideString 型や BSTR 型は、VCL では UnicodeString 型に変更されました。
整数型
ActiveX を C++Builder で使用する際の long 型が int 型に変更されました。
論理型
ActiveX を Delphi で使用する際の WordBool 型が Boolean 型に変更されました。

ImageKit6 VCL
(1)プロパティ名の変更
サムネイルコントロール
ForeColorUp --> ForeColor
PictureUp --> Picture
PictureUpFile --> PictureFile (*)
ThumbSelectCursor --> SelectCursor
移行方法:
[1]dfm ファイルの該当箇所を変更します - (*)以外が対象
[2]pas,cpp ファイルの該当箇所を変更します
(2)仕様変更
サムネイルコントロールの ExifAutoRotate プロパティを追加したことにより、表示される画像の向きが異なる場合があります。
以前の ImageKit と同じ動作とする場合は ExifAutoRotate プロパティを False に設定してください。
サムネイルコントロールの MouseMoveOnThumb イベントの発生条件を変更。詳しくはリファレンスの【ImageKit6/7/8/9 VCL との違い】を参照してください。
(3)その他
詳しくはリファレンスの【ImageKit6 VCL との違い】を参照してください。

ImageKit7 ActiveX
(1)プロパティの削除(ImageKit7 ActiveX → ImageKit10 VCL)
A.イメージキットコントロール
EnableOLEDrag, MouseCursorFile, RectMouseCursorFile
※MouseCursorFile プロパティは Cursor プロパティで代用
※RectMouseCursorFile プロパティは RectCursor プロパティで代用
PanWindow.MouseCursorFile, PanWindow.RectMouseCursorFile
※PanWindow.MouseCursorFile プロパティは PanWindow.Cursor プロパティで代用
※PanWindow.RectMouseCursorFile プロパティは PanWindow.RectCursor プロパティで代用
Scan.DsNameCount, Scan.HalfToneList
B.サムネイルコントロール
EnableOLEDrag, MouseCursorFile
※MouseCursorFile プロパティは Cursor プロパティにて代用
(2)プロパティ名の変更(ImageKit7 ActiveX → ImageKit10 VCL)
A.イメージキットコントロール
BackColor --> Color
Hdc --> Canvas.Handle
Hwnd --> Handle
MouseCursorType --> Cursor
RectMouseCursorType --> RectCursor
Edit.ToolBarMouseCur --> Edit.ToolBarCursor
File --> FileIO
File.JPEG2000NumrResLevel --> FileIO.JPEG2000NumResLevel
Magnifier.Type --> Magnifier.Style
PanWindow.Hwnd --> PanWindow.Handle
PanWindow.MouseCursorType --> PanWindow.Cursor
PanWindow.RectMouseCursorType --> PanWindow.RectCursor
PrintDraw.Hdc --> PrintDraw.Handle
Scan.DsName --> Scan.DataSourceName
Scan.DsNameList --> Scan.DataSourceNameList
Scan.HalfTone --> Scan.Halftone
B.サムネイルコントロール
BackColor --> Color
ForeColorUp --> ForeColor

Hdc --> Canvas.Handle
Hwnd --> Handle
MouseCursor --> SelectCursor
MouseCursorType --> Cursor
PictureUp --> Picture
PictureUpFile --> PictureFile

(3)メソッド名の変更(ImageKit7 ActiveX → ImageKit10 VCL)

イメージキットコントロール

GetPalette --> GetPaletteFromImage
SetPalette --> SetPaletteToImage
Scan.CloseDS --> Scan.CloseDataSource
Scan.Exec --> Scan.Execute
Scan.GetCapEnum --> Scan.GetCapEnumToFloat, Scan.GetCapEnumToString
Scan.GetDSInfo --> Scan.GetDataSourceInfo
Scan.IsOpenDS --> Scan.IsOpenDataSource
Scan.OpenDS --> Scan.OpenDataSource

(4)仕様変更

イメージキットコントロールの **FileIO.ExifAutoRotate** プロパティとサムネイルコントロールの **ExifAutoRotate** プロパティを追加したことにより、表示される画像の向きが異なる場合があります。以前の ImageKit と同じ動作とする場合は **ExifAutoRotate** プロパティを False に設定してください。

サムネイルコントロールの **MouseMoveOnThumb** イベントの発生条件を変更。

(5)その他

詳しくはリファレンスの【ImageKit7/8 ActiveX との違い】【ImageKit7/8/9 ActiveX/VCL との違い】【ImageKit7/8/9/10 ActiveX との違い】を参照してください。

プロパティやメソッドとイベント引数ついて

文字列型

ActiveX の WideString 型や BSTR 型は、VCL では UnicodeString 型に変更されました。

整数型

ActiveX を C++Builder で使用する際の long 型が int 型に変更されました。

論理型

ActiveX を Delphi で使用する際の WordBool 型が Boolean 型に変更されました。

ImageKit7 VCL

(1)プロパティ名の変更

サムネイルコントロール

ForeColorUp --> ForeColor
PictureUp --> Picture
PictureUpFile --> PictureFile (*)
移行方法:
[1]dfm ファイルの該当箇所を変更します - (*)以外が対象
[2]pas,cpp ファイルの該当箇所を変更します

(2)仕様変更

イメージキットコントロールの **FileIO.ExifAutoRotate** プロパティとサムネイルコントロールの **ExifAutoRotate** プロパティを追加したことにより、表示される画像の向きが異なる場合があります。以前の ImageKit と同じ動作とする場合は **ExifAutoRotate** プロパティを False に設定してください。

サムネイルコントロールの **MouseMoveOnThumb** イベントの発生条件を変更。詳しくはリファレンスの【ImageKit6/7/8/9 VCL との違い】を参照してください。

(3)その他

詳しくはリファレンスの【ImageKit7 VCL との違い】【ImageKit7/8/9 ActiveX/VCL との違い】を参照してください。

ImageKit8 ActiveX

(1)プロパティの削除(ImageKit8 ActiveX → ImageKit10 VCL)

A.イメージキットコントロール

EnableOLEDrag, MouseCursorFile, RectMouseCursorFile
※MouseCursorFile プロパティは Cursor プロパティで代用
※RectMouseCursorFile プロパティは RectCursor プロパティで代用
PanWindow.MouseCursorFile, PanWindow.RectMouseCursorFile
※PanWindow.MouseCursorFile プロパティは PanWindow.Cursor プロパティで代用

※PanWindow.RectMouseCursorFile プロパティは PanWindow.RectCursor プロパティで代用
Scan.DsNameCount, Scan.HalfToneList

B.サムネイルコントロール
EnableOLEDrag, MouseCursorFile
※MouseCursorFile プロパティは Cursor プロパティにて代用

(2)プロパティ名の変更(ImageKit8 ActiveX → ImageKit10 VCL)

A.イメージキットコントロール
BackColor --> Color
Hdc --> Canvas.Handle
Hwnd --> Handle
MouseCursorType --> Cursor
RectMouseCursorType --> RectCursor
Edit.ToolBarMouseCur --> Edit.ToolBarCursor
File --> FileIO
File.JPEG2000NumrResLevel --> FileIO.JPEG2000NumResLevel
Magnifier.Type --> Magnifier.Style
PanWindow.Hwnd --> PanWindow.Handle
PanWindow.MouseCursorType --> PanWindow.Cursor
PanWindow.RectMouseCursorType --> PanWindow.RectCursor
PrintDraw.Hdc --> PrintDraw.Handle
Scan.DsName --> Scan.DataSourceName
Scan.DsNameList --> Scan.DataSourceNameList
Scan.HalfTone --> Scan.Halftone

B.サムネイルコントロール
BackColor --> Color
ForeColorUp --> ForeColor
Hdc --> Canvas.Handle
Hwnd --> Handle
MouseCursorType --> Cursor
SelectMouseCursorType --> SelectCursor
PictureUp --> Picture
PictureUpFile --> PictureFile

(3)メソッド名の変更(ImageKit8 ActiveX → ImageKit10 VCL)

イメージキットコントロール
GetPalette --> GetPaletteFromImage
SetPalette --> SetPaletteToImage
Edit.Property --> Edit.ShowPropertyDialog
Scan.CloseDS --> Scan.CloseDataSource
Scan.Exec --> Scan.Execute
Scan.GetCapEnum --> Scan.GetCapEnumToFloat, Scan.GetCapEnumToString
Scan.GetDSInfo --> Scan.GetDataSourceInfo
Scan.IsOpenDS --> Scan.IsOpenDataSource
Scan.OpenDS --> Scan.OpenDataSource

(4)仕様変更

イメージキットコントロールの **FileIO.ExifAutoRotate** プロパティとサムネイルコントロールの **ExifAutoRotate** プロパティを追加したことにより、表示される画像の向きが異なる場合があります。以前の ImageKit と同じ動作とする場合は **ExifAutoRotate** プロパティを False に設定してください。

サムネイルコントロールの **MouseMoveOnThumb** イベントの発生条件を変更。

(5)その他

詳しくはリファレンスの【ImageKit7/8 ActiveX との違い】【ImageKit7/8/9 ActiveX/VCL との違い】【ImageKit7/8/9/10 ActiveX との違い】【ImageKit8/9/10 ActiveX との違い】を参照してください。

プロパティやメソッドとイベント引数ついて

文字列型

ActiveX の WideString 型や BSTR 型は、VCL では UnicodeString 型に変更されました。

整数型

ActiveX を C++Builder で使用する際の long 型が int 型に変更されました。

論理型

ActiveX を Delphi で使用する際の WordBool 型が Boolean 型に変更されました。

ImageKit8 VCL

(1)プロパティ名の変更

サムネイルコントロール

ForeColorUp --> ForeColor

PictureUp --> Picture

PictureUpFile --> PictureFile (*)

移行方法:

[1]dfm ファイルの該当箇所を変更します - (*)以外が対象

[2]pas,cpp ファイルの該当箇所を変更します

(2)列挙型の識別子の変更

TVIkScanUnit

vikScanMillimeters --> vikScanMillimeter

移行方法:

pas,cpp ファイルの該当箇所を変更します

(3)仕様変更

イメージキットコントロールの **FileIO.ExifAutoRotate** プロパティとサムネイルコントロールの **ExifAutoRotate** プロパティを追加したことにより、表示される画像の向きが異なる場合があります。以前の ImageKit と同じ動作とする場合は **ExifAutoRotate** プロパティを False に設定してください。

サムネイルコントロールの **MouseMoveOnThumb** イベントの発生条件を変更。詳しくはリファレンスの【ImageKit6/7/8/9 VCL との違い】を参照してください。

(4)その他

詳しくはリファレンスの【ImageKit7/8/9 ActiveX/VCL との違い】【ImageKit8 ActiveX/VCL との違い】を参照してください。

ImageKit9/10 ActiveX

(1)プロパティの削除(ImageKit9/10 ActiveX → ImageKit10 VCL)

A.イメージキットコントロール

EnableOLEDrag, MouseCursorFile, RectMouseCursorFile

※MouseCursorFile プロパティは Cursor プロパティで代用

※RectMouseCursorFile プロパティは RectCursor プロパティで代用

PanWindow.MouseCursorFile, PanWindow.RectMouseCursorFile

※PanWindow.MouseCursorFile プロパティは PanWindow.Cursor プロパティで代用

※PanWindow.RectMouseCursorFile プロパティは PanWindow.RectCursor プロパティで代用

Scan.DsNameCount, Scan.HalfToneList

B.サムネイルコントロール

EnableOLEDrag, MouseCursorFile

※MouseCursorFile プロパティは Cursor プロパティにて代用

C.プレイコントロール

MouseCursorFile

※MouseCursorFile プロパティは Cursor プロパティにて代用

D.プレビューコントロール

MouseCursorFile

※MouseCursorFile プロパティは Cursor プロパティにて代用

E.レコードコントロール

MouseCursorFile

※MouseCursorFile プロパティは Cursor プロパティにて代用

(2)プロパティ名の変更(ImageKit9/10 ActiveX → ImageKit10 VCL)

A.イメージキットコントロール

BackColor --> Color

EnableTouch --> Touch

Hdc --> Canvas.Handle

HWND --> Handle

MouseCursorType --> Cursor

RectMouseCursorType --> RectCursor

Edit.ToolBarMouseCur --> Edit.ToolBarCursor

File --> FileIO

File.JPEG2000NumrResLevel --> FileIO.JPEG2000NumResLevel

Magnifier.Type --> Magnifier.Style

PanWindow.Hwnd --> PanWindow.Handle

PanWindow.MouseCursorType --> PanWindow.Cursor
PanWindow.RectMouseCursorType --> PanWindow.RectCursor
PrintDraw.Hdc --> PrintDraw.Handle
Scan.DsName --> Scan.DataSourceName
Scan.DsNameList --> Scan.DataSourceNameList
Scan.HalfTone --> Scan.Halftone

B.サムネイルコントロール
BackColor --> Color
EnableTouch --> Touch
ForeColorUp --> ForeColor
Hdc --> Canvas.Handle
HWND --> Handle
MouseCursor --> SelectCursor
MouseCursorType --> Cursor
PictureUp --> Picture
PictureUpFile --> PictureFile

C.プレイコントロール
BackColor --> Color
HWND --> Handle
MouseCursorType --> Cursor
VideoHeight,VideoWidth --> VideoSize

D.プレビューコントロール
BackColor --> Color
HWND --> Handle
MouseCursorType --> Cursor
VideoHeight,VideoWidth --> VideoSize

E.レコードコントロール
BackColor --> Color
HWND --> Handle
MouseCursorType --> Cursor
VideoHeight,VideoWidth --> VideoSize

(3)メソッド名の変更
[1]ImageKit9/10 ActiveX → ImageKit10 VCL
イメージキットコントロール
GetPalette --> GetPaletteFromImage
SetPalette --> SetPaletteToImage
Edit.Property --> Edit.ShowPropertyDialog
Scan.CloseDS --> Scan.CloseDataSource
Scan.Exec --> Scan.Execute
Scan.GetCapEnum --> Scan.GetCapEnumToFloat, Scan.GetCapEnumToString
Scan.GetDSInfo --> Scan.GetDataSourceInfo
Scan.IsOpenDS --> Scan.IsOpenDataSource
Scan.OpenDS --> Scan.OpenDataSource

[2]ImageKit10 ActiveX → ImageKit10 VCL
イメージキットコントロール
File.PDF.End --> FileIO.PDF.Finish

(4)仕様変更(ImageKit9 ActiveX → ImageKit10 VCL)
イメージキットコントロールの **FileIO.ExifAutoRotate** プロパティとサムネイルコントロールの **ExifAutoRotate** プロパティを追加したことにより、表示される画像の向きが異なる場合があります。以前の ImageKit と同じ動作とする場合は **ExifAutoRotate** プロパティを False に設定してください。
サムネイルコントロールの **MouseMoveOnThumb** イベントの発生条件を変更。

(5)その他
詳しくはリファレンスの【ImageKit7/8/9/10 ActiveX との違い】【ImageKit7/8/9 ActiveX/VCL との違い】【ImageKit8/9/10 ActiveX との違い】【ImageKit9/10 ActiveX との違い】を参照してください。

プロパティやメソッドとイベント引数ついて
文字列型
ActiveX の WideString 型や BSTR 型は、VCL では UnicodeString 型に変更されました。
整数型

ActiveX を C++Builder で使用する際の long 型が int 型に変更されました。

論理型

ActiveX を Delphi で使用する際の WordBool 型が Boolean 型に変更されました。

ImageKit9 VCL

(1)仕様変更

イメージキットコントロールの FileIO.ExifAutoRotate プロパティとサムネイルコントロールの ExifAutoRotate プロパティを追加したことにより、表示される画像の向きが異なる場合があります。以前の ImageKit と同じ動作とする場合は ExifAutoRotate プロパティを False に設定してください。

サムネイルコントロールの MouseMoveOnThumb イベントの発生条件を変更。詳しくはリファレンスの【ImageKit6/7/8/9 VCL との違い】を参照してください。

(2)その他

詳しくはリファレンスの【ImageKit7/8/9 ActiveX/VCL との違い】【ImageKit9 ActiveX/VCL との違い】を参照してください。

索引

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

く

こ

さ

し

す

せ

て

と

ね

の

は

ひ

ImageKit10 VCL

コントロールリファレンス

発行 株式会社ニュートン

〒940-0076
新潟県長岡市本町 2-2-15 シャングリラ本町 1F
TEL 0258-86-6954
FAX 0258-86-6964
http://www.newtone.co.jp/