IFoerster Development

# FoxBurner SDK Guide

# Contents

# FoxBurner SDK Guide

## Introduction


**FoxBurner SDK** extends your company to give you the ability to quickly and easily create professional applications that have the latest disc recording features that are seen in such popular products.

**Supported Envrionments**
Visual Basic 6
Visual Studio 6
Visual Studio 2002-2012
Borland Delphi & C++Builder
embarcadero RAD Studio (All versions)
QT-Designer

**Supported Operating Systems**

Windows NT 4.0, 2000, XP
Windows Vista, Windows 7, Windows 8
Windows Server 2003 (R2), Windows Server 2008 (R2), Windows Server 2012
MacOS (Intel)
Linux

Development may perform improvements and/or changes in the product(s) and/or the program(s) described in this paper at any time without notice.

Any references in this document to non-IFoerster Development web sites are provided for convenience only and do not in any manner serve as an endorsement of those web sites. The materials affiliated with other web sites are not part of the materials describing the IFoerster Development products and use of those materials will be done at your own risk.

IFoerster Development may have patents or pending patent applications covering subject matters described in this document. The furnishing of this document does not give you any license to these patents.

## Migrate from older FoxBurner SDK Versions

The FoxBurner SDK 6.x and above was completely redeveloped to support the latest functions and operating systems. The logic behind the SDK and its functions is the same as before, but the interface has changed in many ways.

Therefore, it is not possible to migrate with one step from the older FoxBurner SDK to this new FoxBurner SDK 6.x and above. You need to adapt your code to the new interface.

We suggest to use the Unicode version of the FoxBurner SDK to give your software the capability to work with Asian languages. The multi-byte version of the FoxBurner SDK is for backward compatibility only.

## FoxBurner SDK Versions

FoxBurner offers a number a different license models that fit your individual needs. The following table lists the license models and the developers they are designed for.

To see all the features that come with a specific license, go to the comparison page on our website. This overview helps you to find the perfect license for your individual needs.

Although, all license models are subscription licenses, you can use the SDK after the subscription has expired! You simply do not get any updates and support anymore, in case you do not renew your subscription.

**FoxBurner SDK Limited Edition**

The FoxBurner SDK Limited Edition should be used by developers who just need one operating system license. This license is dedicated to one developer and the distribution is royalty free.

**FoxBurner SDK XPlatform Edition**

This is the license to use the FoxBurner SDK on all available operating systems. You will get 3 license keys. This license is dedicated to one developer and the distribution is royalty free.

**FoxBurner SDK Server Edition**

This license is for networking or data center projects. This license is dedicated to one one CPU server and one developer. The number of client applications is unlimited.
The Server Edition is an AddOn license. You will need a Single Edition license to obtain the Server Edittion.

**FoxBurner SDK Source Edition**

This license contains the source code of the FoxBurner SDK. This license will be delivered only physical with UPS. This license is dedicated to one developer.
The Source Edition is an AddOn license. You will need a XPlatform license to obtain the Source Edition.

**FoxBurner SDK Enterprise Edition**

This license is for publishers/companies with a high amount of software products or spin-off software products. This license is not limited to one software product. The distribution is royalty free.

## Technical Support

**E-mail Support**

All licenses contain a 30 days priority support with e-mail and our forum. You will not get Phone or Skype support with this ground support type.

**Forum Support**

IFoerster Development offers a moderated support forum. Here you can get help from us or from our existing customers. Enterprise license discussions are not public.

**Support Plan**

IFoerster Development offers a support plan. If you obtain a support plan, you will get Phone and Skype support. Also the reaction time increase to 24 hours.

# SDK License Agreement

Please review the following license agreement before installing or using IFoerster FoxBurner SDK, any individual IFoerster software component or other software product, and/or all related materials. If you agree to the terms herein then you must click the "YES" button during the software installation process as confirmation before you are permitted to use this software and related documents.

**Definitions:**

This FoxBurner SDK End-User Software License Agreement ("EULA") is a legal agreement between you (either as an individual user, corporation or single entity) and Ingo Foerster ("IFoerster") for the IFoerster FoxBurner SDK or individual software component which includes computer software, and may include associated media, printed materials, and "online" or electronic documentation ("SOFTWARE PRODUCT" or "SOFTWARE"). By exercising your rights to install the SOFTWARE PRODUCT, you agree to be bound by the terms of this End User License Agreement ("EULA"), including the limitations and warranty disclaimers. If you do NOT agree to the terms of this EULA, you may not install nor use the SOFTWARE PRODUCT; If this is the case, please uninstall the SOFTWARE PRODUCT from your system immediately, and destroy all copies of the SOFTWARE PRODUCT and all of its component parts, source code, associated documentation, and related materials.

**SOFTWARE PRODUCT LICENSE:**

The SOFTWARE PRODUCT is protected by copyright laws and international copyright treaties, as well as other intellectual property laws and treaties.

**1. GRANT OF LICENSE.**

This is a license agreement, and NOT an agreement for sale. IFoerster retains ownership of the copy of THE SOFTWARE in your possession, and all copies you may be licensed to make. IFoerster retains all rights not expressly granted to you in this LICENSE. IFoerster hereby grants to you, and you accept, a non-exclusive, non-transferable license to use, copy and modify THE SOFTWARE only as authorized below.

Provided that you have accepted these terms contained herein by clicking the "YES" button during the software installation process, this EULA grants you the following rights:

**Installation and Use:**

DEVELOPER LICENSE: For single-version licenses, you are granted a license as a single developer (individual) to the particular licensed version of the SOFTWARE PRODUCT. For subscription-based licenses, you are granted a license as a single developer (individual) to all versions of the particular product released during the period of the subscription. You are further licensed to distribute the SOFTWARE PRODUCT royalty-free on an unlimited number of workstations within a single organization, provided that these applications are developed solely by you. At no time may the SOFTWARE PRODUCT be used by more than one individual at the same time for development purposes – nor may the SOFTWARE PRODUCT be distributed for use with applications other than those developed by you.

You are required to reasonably ensure that the SOFTWARE PRODUCT is not reused by or with any application other than those with which you distribute it. For example, if you install the FoxBurner SDK along with a packaged application on a customer's server, that customer is not permitted to use the SOFTWARE PRODUCT independent of your application, and must be informed as such.

FOR ALL LICENSES: In no case shall you rent, lease, lend, redistribute nor re-license THE SOFTWARE PRODUCT or any related source code to a 3rd party individual or entity, except as outlined above. In no case shall you grant further redistribution rights for THE SOFTWARE PRODUCT to the end-users of your solution. UNDER NO CIRCUMSTANCES MAY THE SOURCE CODE OR THE COMPILED PRODUCT BE USED AS THE BASIS FOR CREATING A PRODUCT THAT CONTAINS THE SAME, OR SUBSTANTIALLY THE SAME, FUNCTIONALITY AS ANY IFOERSTER DEVELOPMENT PRODUCT.

**2. DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS.**

Termination: Without prejudice to any other rights, IFoerster may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts, source code, associated documentation, and related materials.

**3. COPYRIGHT.**

All title and copyrights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text, and "applets" incorporated into the SOFTWARE PRODUCT), the accompanying printed materials, and any copies of the SOFTWARE PRODUCT are owned by IFoerster. The

SOFTWARE PRODUCT is protected by German copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE PRODUCT like any other copyrighted material.

**4. LIMITED WARRANTY.**

NO WARRANTIES. IFoerster expressly disclaims any warranty for the SOFTWARE PRODUCT. The SOFTWARE PRODUCT and any related documentation is provided “as is” without warranty of any kind, either expressed or implied, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. The entire risk arising out of use or performance of the SOFTWARE PRODUCT remains with you.

**5. LIMITATION OF LIABILITY.**

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall IFoerster or its distributors be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use this IFoerster product (THE SOFTWARE PRODUCT) and related materials, even if IFoerster has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

# Extras

## Blu-ray Video

The FoxBurner SDK supports the writing of Blu-ray Video Disks. You can write it on BD-R, BD-RE and BD-XL. The writing on DVD for AVCHD is supported, too. Following Blu-ray specials are implemented into the SDK:

Fixed Start LBA according to the Blu-ray specification.

All files inside the Stream folder will become the UDF real-time streaming attribute.

With the FoxBurner SDK custom sorting feature you can place the files on the final disc according to your needs.

## BD-XL and M-Disc

The FoxBurner SDK support the reading an writing of BD-XL disc types. Also the writing and reading of the M-Disc format is supported.

## Custom sorting feature

With the new version of the FoxBurner SDK, it comes with a new custom sorting feature. It allows you to place the files on the disc wherever you want. So you can place sensitive data in the inner part of the disc, because it is more safe than the outer area. Or you can sort a Blu-ray Video disc according to your needs, as sample for 3D discs.

**How the sorting works**
You can sort according to different parameters, like sorting by file name. The Data sample that is included in the SDK demonstrates the use of sorting by implementing the file priority feature. Each file is assigned a priority value in the range 0-100, and by default all files have equal priority of 0. The files with greater priority are placed closer to the center of a disc, thus will be more protected from scratches and fingerprints. To store the priority value the sample uses pUserParam member variable of the SFileEntry structure that is associated with each file. The pUserParam member variable is dedicated to store any user-defined data that user wants to associate with each file or directory added to the project.

To make use of new sorting feature you should define a special comparison callback function that takes two files as parameters and should return BS_TRUE if the first file should be placed before the second file into the disc image, otherwise it should return BS_FALSE.

```
BS_BOOL CompareFilesForArrangementEvent(
const SFileEntry * pFile1,
const SFileEntry * pFile2,
void * pUserData
);
```

The Data sample defines the callback as static function in the class CDataMFCDlg:

```
BS_BOOL CDataMFCDlg::OnCompareFilesForArrangement(

const SFileEntry *pFile1,

const SFileEntry *pFile2,

void *pUserData

){
// We are using pUserParam of each file to store the file priority marker.
// Files with higher priority will be placed first in the resulting image.
return pFile1->pUserParam > pFile2->pUserParam;
}
```

Then you should let the SDK know about your callback using SetCompareFilesForArrangementEventCallback() function:

```
SetCompareFilesForArrangementEventCallback(CDataMFCDlg::OnCompareFilesForArrangement, this);
```

The callback will be called for each file. To be precise the number of calls will be N*log2(N) where N is the number of files. The callback is automatically called by the SDK during the preparation of the disc image before burning.

**Note:**
If you do not set the callback, then the SDK will sort the files according their names.

## Options Ruleset

The following table lists the available options for different project types.

| | |
|---|---|
| 1 = BS_BURNER_DATA | 2 = BS_BURNER_UDFDVD |
| 3 = BS_BURNER_ISOUDF | 4 = BS_BURNER_BLURAY |
| 5 = BS_BURNER_AUDIO | 6 = BS_BURNER_MIXEDMODE |
| 7 = BS_BURNER_VIDEODVD | 8 = BS_BURNER_VCD |
| 9 = BS_BURNER_SVCD | 10 = BS_BURNER_CUE |
| 11 = BS_BURNER_RAW | |

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Label** | X | X | X | X | | X | X | X | X | | |
| **Write Method** | | | | | | | | | | | |
| **Joliet** | X | | X | | | X | | | | | |
| **Boot** | X | | X | | | | | | | | |
| **Finalize** | X | X | X | X | | | | | | | |
| **Test Burning** | X | X | X | X | X | X | X | X | X | X | X |
| **OPC** | X | X | X | X | X | X | X | X | X | X | X |
| **Verify** | X | X | X | X | | X | X | | | | |
| **Eject** | X | X | X | X | X | X | X | X | X | X | X |
| **UnderrunProtection** | X | X | X | X | X | X | X | X | X | X | X |
| **UDF** | | X | X | X | | | | | | | |
| **IsoEx** | X | | X | | | X | X | | | | |
| **Compression** | X | | | | | | | | | | |
| **Encryption** | X | | | | | | | | | | |

If „GetOptions“ is called all options that are not available in the context of a certain project are still marked as „BS_NDEF“.

## User Rights / Burn Non-Admin

The FoxBurner SDK is developed according the User Rights / Restrictions of the Windows Operating Systems. Following you can find a short introducing of the FoxBurner User Admission Service, a Service that works according to ACL Rules on a Windows System.
Cause FoxBurner SDK is a high professional SDK there is "No" general solution available. Most of this "General solutions" are not conform with Company rulesets. You can download a extended letter at: http://www.pixbyte.com/download/FoxUas.pdf.

**The User Admission Service**

FoxUas.exe User Admission Service, the service itself. The service must be installed by an administrator and needs to be run in an admin context.

FoxUas.dll User Admission Service DLL, a dll needed by the user admission service.

**Parameters:**

FoxUas - [ install | auto | stop | remove] GroupName Drive

**Sample how to set up the service**

The service is set up and started by calling run.bat that contains the following call:

FoxUas -install -auto DVDGroup d:\

"DVDGroup" is the group with burning rights; whereas "d:\" is the designated drive letter of the recording device.

**Sample how to stop and remove the service**

The service is stopped and removed by calling stop.bat that contains the following two lines:

FoxUas -stop

FoxUas -remove

**The ACL User Manager**

The UserSelection.exe is the user administration tool. The admin can set up and remove burning rights to users by assigning them to the burn group. All users of this group have burning rights and are allowed to record data as long as the service is running and until they are removed from the group again.

## Windows .Net Usage

2005 - .NET Framework 2.0

2008 - .NET Framework 3.5

2010 - .NET Framework 4.0

**Distrinution:**

**Windows 32 bit:**

**FoxSDK32Net05.dll**
32-bit FoxSDK core library for .NET linked with Visual Studio 2005 runtime library.

**FoxSDK32Net08.dll**
32-bit FoxSDK core library for .NET linked with Visual Studio 2008 runtime library.

**FoxSDK32Net10.dll**
32-bit FoxSDK core library for .NET linked with Visual Studio 2010 runtime library.

**Windows 64 bit:**

**FoxSDK64Net05.dll**
64-bit FoxSDK core library for .NET linked with Visual Studio 2005 runtime library.

**FoxSDK64Net08.dll**
64-bit FoxSDK core library for .NET linked with Visual Studio 2008 runtime library.

**FoxSDK64Net10.dll**
64-bit FoxSDK core library for .NET linked with Visual Studio 2010 runtime library.

**Windows UI:**

**FoxSDKDotNetGUI_2005.dll**
FoxSDK standard UI dialogs for .NET linked with FoxSDK32Net05.dll.

**FoxSDKDotNetGUI_2008.dll**
FoxSDK standard UI dialogs for .NET linked with FoxSDK32Net08.dll.

**FoxSDKDotNetGUI_2010.dll**
FoxSDK standard UI dialogs for .NET linked with FoxSDK32Net10.dll.

## Windows Usage

On Windows, there is a Unicode and a Multibyte build available. The Unicode build is always marked with a “U” in the name.
The “C” and “W” in the names identify the platform. “W” = Windows, means with MFC compiled. “C” is cross platform and compiled with Fox-Toolkit.

**Installation:**

FoxBurner SDK is installed to: C:\ProgramFiles\ FoxBurner SDK 6xx

**Distribution:**

**MFC Version:**

**Windows 32 bit:**

**FoxSDK32w.dll**
32-bit FoxSDK core library with MFC UI, using multi-byte character set.

**FoxSDKU32w.dll**
32-bit FoxSDK core library using with MFC UI, Unicode character set.

**Windows 64 bit:**

**FoxSDK64w.dll**
64-bit FoxSDK core library with MFC UI, using multi-byte character set.

**FoxSDKU64w.dll**
64-bit FoxSDK core library with MFC UI, using Unicode character set.

**Fox –Toolkit Version:**

**Windows 32 bit:**

**FoxSDK32c.dll**
32-bit FoxSDK core library with FOXToolkit UI, using multi-byte character set.

**FoxSDKU32c.dll**
32-bit FoxSDK core library with FOXToolkit UI, using Unicode character set.

**Windows 64 bit:**

**FoxSDK64c.dll**
64-bit FoxSDK core library with FOXToolkit UI, using multi-byte character set.

**FoxSDKU64c.dll**
64-bit FoxSDK core library with FOXToolkit UI, using Unicode character set.

**Fox-Toolkit:**

**FoxSDKFXToolkit.dll**
FOXToolkit for FoxSDK library using multi-byte character set.

**FoxSDKFXToolkitU.dll**
FOXToolkit for FoxSDK library using Unicode character set.

**Windows Extras:**

**Windows 32 bit:**

**FoxEncLib32.dll**
32-bit audio encoding library.

**FoxPlayer32.dll**
32-bit audio decoding library using multi-byte character set.

**FoxPlayerU32.dll**
32-bit audio decoding library using Unicode character set.

**FoxWmaPlugin32.dll**
32-bit plugin for FoxPlayer library to decode WMA files, using multi-byte character set.

**FoxWmaPluginU32.dll**
32-bit plugin for FoxPlayer library to decode WMA files, using Unicode character set.

**Windows 64 bit:**

**FoxEncLib64.dll**
64-bit audio encoding library.

**FoxPlayer64.dll**
64-bit audio decoding library using multi-byte character set.

**FoxPlayerU64.dll**
64-bit audio decoding library using Unicode character set.

**FoxWmaPlugin64.dll**
64-bit plugin for FoxPlayer library to decode WMA files, using multi-byte character set.

**FoxWmaPluginU64.dll**
64-bit plugin for FoxPlayer library to decode WMA files, using Unicode character set.

## MAC Usage

The FoxBurner SDK work under the latest MAC Intel Architecture. The SDK contains a set of cross platform samples, developed with Fox-Toolkit and one XCode project.

**Installation:**

Sample files are installed to: /Users/Shared/FoxSDK/Samples

Library files are installed to: /usr/local/lib

Documentation is installed to:

**Distribution:**

**libFoxSDK.dylib**
FoxSDK core library.

**libFoxSDKFXToolkit.dylib**
FOXToolkit for FoxSDK.

**libFoxEncLib.dylib**
Audio encoding library.

**libFoxPlayer.dylib**
Audio decoding library.

**Compile:**

**Compiler:**

Compiler is GCC. Use Xcode as development environment.

## Linux Usage

The FoxBurner SDK work under the latest Linux Intel Architecture. The SDK contains a set of cross platform samples, developed with Fox-Toolkit.

**Installation:**

Sample files are installed to: /usr/share/FoxSDK/Samples

Library files are installed to: /usr/bin

Documentation is installed to: /usr/share/doc/FoxSDK

**Distribution:**

libFoxSDK.so - FoxSDK core library.

libFoxSDKFXToolkit.so - FOXToolkit for FoxSDK.

libFoxEncLib.so - Audio encoding library.

libFoxPlayer.so - Audio decoding library.

**Compile:**

**Compiler:**

Compiler is GCC. Use Xcode as development environment.

## Delphi & Firemonkey Usage

The FoxBurner SDK is an ultra-modern and high-performance CD/DVD recording SDK (Software Development Kit). It is not a visual software (Non-visual). Therefore, for a program writer it doesn't matter if it's going to be used with VCL or FireMonkey.

But in order to guarantee that the software will function on various systems and platforms, no platform-specific APIs should be employed. In case you cannot stay away from platform-specific APIs for some reason, you must use conditional compilation and carry out this operation independently for each platform that is supported. For example:

```
{$IFDEF MSWINDOWS}
DoSomethingWindowWay;
{$ENDIF MSWINDOWS}
{$IFDEF OSX}
DoTheSameOSXWay;
{$ENDIF OSX}
```

If you don't want to write these types of codes, then you have to try your best to use the Delphi runtime library as much as you can, rather than platform-specific calls. The RTL will deal with these kinds of situations effectively on supported platforms, free of charge.

Delphi XE2 defines many symbols which you can make use of in {$IFDEF} constructs and which would help you while writing multi-platform codes. For example, OSX is identified when you compile the code for Mac OS X platform, and WIN64 is defined when you compile the code to Windows x64 native code.

For more details take a look at: http://docwiki.embarcadero.com/RADStudio/en/Conditional_compilation_(Delphi)

The real challenge that arises while porting the FoxBurnerSDK software to make it work with many versions of Delphi and on several platforms lies in the fact that FoxBurner SDK is made to work with different character sets, based upon the Delphi version as well as the platform

- On Delphi 7 (Windows only): The Software employs a multi-byte character set and multi-byte version of FoxSDK DLL. The reason being Delphi 7 doesn't support Unicode. SizeOf(Char) = 1;
- From Delphi 2009 to Delphi XE on Wi32/Win64 platforms : The Software uses Unicode character set and Unicode version of FoxSDK DLL. These Delphi versions support Unicode. SizeOf(Char)=2;
- In Delphi XE2 on Mac OS X platform : The software makes use of a multi-byte character set, since FoxSDK library for Mac OS X is compiled only in a multi-byte version.

This task however was sorted out with the introduction of new character and string types, which have different internal representation based on the platform as well as depending on whether their current version of Delphi is Unicode or not.

```
{$IFDEF MSWINDOWS}
FoxSDKChar    = Char;
PFoxSDKChar   = PChar;
FoxSDKString  = String;
{$ELSE MSWINDOWS}
FoxSDKChar    = AnsiChar;
PFoxSDKChar   = PAnsiChar;
FoxSDKString  = AnsiString;
{$ENDIF MSWINDOWS}
```

These FoxSDK-specific character and string types are usually applied internally all through the program code for string manipulation.

Another issue that arose concerned the FoxSDK license keys. They are distinct for each of the supported platforms. Earlier, the VCL software had a single License Key property which was used to store the license key for Windows. Now the software has two distinct properties, i.e. License Key For Mac OSX and License Key For Windows, which are used to set the license key for both the supported platforms. Thus, now setting up the old License Key property results in setting the License Key For Windows Property only.

## BootImage

The FoxBurner SDK enables you to write bootable data CD/DVD by using your own boot images. The SDK supports all currently used image types (as long as they are compatible with your systems).

A boot CD/DVD can only be created while working with data and ISOUDF projects..

The following functions and options are available in the context of boot images:

**Boot CD/DVD Options**

These are the basic settings for a boot CD/DVD. For functions and properties, please refer to the options section of the SDK help file.

**Extended Settings:**

By setting the Extended BootInfos you can specify the boot image.  More information can be found in boot image section of the SDK help file.

The settings are for professional users only. Please make no changes if you are not familiar with the specifications. This settings are part of the El Torito Specs, you can download them here: "El Torito" Specification

Boot images can be built for or extracted from several systems (like drdos, Linux, and others). These images have a certain format/structure that should not be altered in any way. There are several tools available to create a boot image. One of them that is easy to use is WinImage (http://www.winimage.com/).

The FoxBurner SDK package includes a boot image on the basis of OpenDos (Calderra)*. The latest version of OpenDos may not be distributed freely. If you need to distribute this boot image, get in contact with the copyright owner and ask for the current terms and conditions to acquire a license. (http://www.drdos.com).

You can find the BootImage in the installation folder: „ FoxBurner SDK >AddOns>BootImages"

*) You can use the provided image for testing only. If you want to include this image in your projects and distribute it with your application, get in contact with the copyright owner to prevent a possible copyright violation.

## Multi-Device

The FoxBurner SDK now supports burning on several devices simultaneously. To activate multi-device burning, the function “AddDevice” instead of “SetDevice” needs to be called. You can add n devices to the internal device list. Die Function SetDevice setzt die interne Liste zurück. Sprich nach dem Aufruf von AddDevice sollten Sie nur SetDevice aufrufen um die Liste bzw. Das Multidevice brennen zurückzusetzen.

If you have called „AddDevice“, you have to specify the devices by addressing the index of a certain device while working with device related functions. To determine the number of devices, call the function „GetDeviceCount.“

If more than one device has been added to the internal list, the function „GetDevice“ will return a mask of drives.

**Example :**

```
SetBurnDevice(“C”);
GetBurnDevice = “C”;

AddDevice(“C”);
AddDevice(“D”);

GetBurnDevice = “CD”;
```

If you have added more than one device to the internal list, determining the writing speeds will return an array of burning speeds that are supported by all devices. E.g. you have added one device with max speeds 8x and one device with 16x the overall max recording speed is 8 times.

## Disk FileSystems

The FoxBurner SDK supports different Disk FileSystems and combinations “Brdiges”. Please see the following table with the supported File Systems:

| File System | Specials | Disks |
|---|---|---|
| ISO 9660 | ISO 9660 Specification | CD /DVD |
| Level 1 | Name format is 8.3. Available chars: only capital letters and underscore.<br>File extentio length is limited to 3 chars.<br>Directory can't has extention.<br>File size limit is 2 GB. | |
| Level 2 | Max name length is 31 chars.<br>File size limit is 2 GB. | |
| Level 3 | Max name length is 128 chars.<br>No file size limit. | |
| Joliet | Max name length 64 chars<br>Max directory name length 120 chars<br>Unicode allowed<br>Upper/Lower caps allowed<br>File size limit 2 GB | |
| Romeo | Max name length is 128 chars.<br>Lower case chars are allowed in file names. | |
| Rockridge | Rockridge Intercharge Protocol | CD /DVD |
| UDF 1.02 | UDF Revision 1.02 | CD /DVD |
| UDF 1.50 | UDF Revision 1.50 | CD /DVD |
| UDF 2.00 | UDF Revision 2.00 | CD /DVD |
| UDF 2.01 | UDF Revision 2.01 | CD /DVD |
| UDF 2.50 | UDF Revision 2.50 | CD /DVD / BD |
| UDF 2.60 | UDF Revision 2.60 | CD /DVD / BD |

## Write Audio CD

The  FoxBurner SDK includes an own decoder to burn audio files in the format.

OGG, WAV, MP3, WMA, AAC, ASF, VQF, M4A, FLA

The decoder is also able to give information about duration and to playback the audio files.

The FoxBurner SDK also includes an undocumented Tag reading library that you can use to read and write tags to/from : WMA; MP3, OGG, AAC and M4A. To use the TAG library please contact our support.

# Writing

The RAW Writing was added to the SDK to support own image files to burn. Also you can burn existing Image Files from other software as long you know the structure of the files.

The RAW Sample in the SDK is using the FoxBurner ImageConverter to analyze image files. This was added to give you a tutorial how the image structure has to be added to the FoxBurner SDK.

**Important:**
The FoxBurner ImageConverter is not part of the FoxBurner SDK. You are not allowed to redistribute this file without an additional agreement.

## Work with the RAW Writing

With RAW project you can burn data from file or you can use callback function to provide data.

To create RAW project use CreateProject(BS_BURNER_RAW).

Available functions:

**AddFile()**

Through AddFile() you can specify from which file to take the data for burning. If the file is not specified, callback function will be used to obtain data.

**SetRAWStructure()**

The SetRAWStructure is designed to specify the structure of the disc and the format of the input data.

SetRAWStructure(struct SRAWTrackToAdd* pTracks, int32 nLength)

**Parameters:**

pTracks
array of structures describing disk tracks

nLength
number of elements in pTracks array

**Special data type**

```
struct SRAWTrackToAdd:
{
        int32 nNumber;
        int32 nIndex;
        int32 nFormat;
        int32 nDataTypeMask;
        int32 nIgnoreDataMask;
        int32 nStartAddress;
        int32 nLength;
        int64 nOffset ;
}
```

**int32 nNumber**
Track number. 0 - Lead-in, 0xAA – lead-out. 1 – first track, 2 – second track ...

**int32 nIndex**
Index inside track. 0 - pre-gap, 1..99 - user data

**int32 nFormat**
Track format: audio, mode1, mode2 ... Shall be one of BS_RTF_* values.

**int32 nDataTypeMask**
Data format. Shall be a combination of BS_RDT_* flags. It is allowed to add an enumeration on time to the list.

**int32 nIgnoreDataMask**
Data to be ignored format. Shall be a combination of BS_RDT_* flags. It is allowed to add an enumeration on time to the list.

**int32 nStartAddress**
Track start address on disc in sectors.

**int32 nLength**
Track length in sectors.

**int64 nOffset**
Offset in the image file in bytes. Not used if burning through callback functions.

## *IgnoreDataMask Information*

The ignore mask is used to specify which of the input data fields should be ignored and generated automatically.

**Example 1.**

Your file consists of sectors of 2352 bytes and contains fields Sync/Header, User Data and ECC/EDC. But you want to burn only. Then you specify:

nDataTypeMask = BS_RDT_SYNC_HEADER | BS_RDT_DATA | EDC_ECC;

nIgnoreDataMask = BS_RDT_SYNC_HEADER | EDC_ECC;

In this case Sync/Header and EDC/ECC will be automatically generated by device or by FoxBurner SDK.

**Example 2.**

(Burning *.iso file.)

SRAWTrackToAdd track;

track.nNumber =1;

track.nIndex = 1;

track.nFormat = BS_RTF_MODE1;

track.nDataTypeMask = BS_RDT_DATA;

track.nIgnoreDataMask = 0;

track.nStartAddress = 0;

track.nLength = SizeOfFile(filename)/2048;

track.nOffset = 0;

### *Lead-in, Lead-out and first pre-gap.*

Descriptors for lead-in, lead-out and pre-gap of the first track are not mandatory but can be specified. If you don't send them they will be generated automatically.

### *Sub channel data.*

When recording sub channel data only one of the flags BS_RDT_SUBCH_ * may be specified.

**BS_RDT_SUBCH_PQ**
User provides 16-byte P and Q in formatted form (see MMC 5). More channels are filled 0.

**BS_RDT_SUBCH_PW**
User provides 96 bytes of raw P-W channels data.

**BS_RDT_SUBCH_RW (CD-TEXT)**
User provides 96 bytes of raw data for P-W channels. Data for P and Q will be ignored and generated automatically.

### *Callback function*

int32 RAWDataEvent(int32 nLBA, void* pBuffer, int32 nBufferSize, int32* nReaded, void* pUserData)

**Parameters:**

**int32 nLBA**
Current logical block address (may be negative)

**void* pBuffer**
Pointer to buffer

**int32 nBufferSize**
Size of buffer in bytes

**int32* nReaded**
Pointer to return number of bytes read

**void* pUserData**
User data

**Return Value**

BS_SDK_ERROR_NO if a read succeeded or an error code is returned if a read failed. If an error is returned the burning process will be terminated.

### *RAW Table*

| | AUDIO | MODE1 | MODE2 FORMLESS | MODE2 FORM1 | MODE2 FORM2 |
|---|---|---|---|---|---|
| SYNC_HEADER | 0 | 16 | 16 | 16 | 16 |
| SUBHEADERS | 0 | 0 | 0 | 8 | 8 |
| DATA | 2352 | 2048 | 2336 | 2048 | 2324 |
| EDC_ECC | 0 | 288 | 0 | 280 | 4 |
| SUBCH_PQ | 16 | 16 | 16 | 16 | 16 |
| SUBCH_PW | 96 | 96 | 96 | 96 | 96 |
| SUBCH_RW | 96 | 96 | 96 | 96 | 96 |

# DiskCopy

The FoxBurner SDK offers two different types of disk copy functions.

## *ImageCreation*

The ImageCreation method creates an image from the target drive. Depending on the source medium an ISO or BIN file will be generated.

To illustrate this feature a small application called UtilitiesMFCU.exe is included in the SDK package.

## *DirectCopy*

By implementing the DirectCopy function you can directly create copies of the source CD/DVD or Blu-ray medium. Simply select the source and target drive and start the process.

To illustrate this feature a small application called UtilitiesMFCU.exe is included in the SDK package.

## *CopyDisk()*

Copy one disc to another. A current reading device (BS_READ_DEVICE) is used as a source device, and a current burn device (BS_BURN_DEVICE) is used as destination device. Other settings are specified using the structure SDiskCopyOptions.
Possible disc copying: CD-> CD, DVD-> DVD, BD-> BD.
If copying is not possible, the function will return error BS_SCSI_ERROR_DISK_30.
Callbacks used during the copy: ProcessEvent, FinalizeEvent, BurnDoneEvent, JobDoneEvent

int32 CopyDisk (SDiskCopyOptions cDiskCopyOptions);

**Parameters:**

SDiskCopyOptions cDiskCopyOptions

**Sepcial Data Type:**

**struct SdiskCopyOptions**

{

uint8 nWriteMethod;

uint8 nReadMode;

BS_BOOL bVerifyAfterBurn;

SReadErrorCorrectionParams cErrorParams;

}

**uint8 nWriteMethod**
The method of recording. Acceptable options for the CD: BS_WM_DAO, BS_WM_DAO96. For DVD and BD discs this field is ignored.

**uint8 nReadMode**
The method of reading. For CD: BS_RM_USERDATA - read only user data.
BS_RM_RAW - sector are read in "raw" mode.
BS_RM_RAW_SUBCHANNEL - sector are read in "raw" mode + reading subchannel data.
For DVD / BD this field is ignored.

**BS_BOOL bVerifyAfterBurn**
BS_TRUE - verify disk after burn, BS_FALSE - do not verify disk after burn

**SReadErrorCorrectionParams cErrorParams**
Read error correction settings

**struct SreadErrorCorrectionParams**

{

BS_BOOL bErrorCorrection;

BS_BOOL bBlankBadSectors;

uint8 nHardwareRetryCount;

uint8 nSoftwareRetryCount;

}

**BS_BOOL bErrorCorrection**
BS_TRUE - Error correction is enabled, BS_FALSE - Error correction is disabled. Sectors with read errors are skipped.

**BS_BOOL bBlankBadSectors**
BS_TRUE - in case of reading errors, the sector will be filled with zeros, BS_FALSE - sector containing error will be written "as is".

**uint8 nHardwareRetryCount**
The number of hardware read retries

**uint8 nSoftwareRetryCount**
The number of software read retries

## *CreateImage()*

Creating a disk image with (possibly) verification. cCreateImageParams - Settings of image creation. NTaskType - Sets the type of task. Possible options:
BS_IMGTASK_CREATE - create an image
BS_IMGTASK_VERIFY - compare existing image with a disk
BS_IMGTASK_CREATE | BS_IMGTASK_VERIFY - create an image with verification.

Callbacks used during image creation and verification: ProcessEvent, BurnDoneEvent, JobDoneEvent, StartVerifyEvent, VerifyFileEvent, VerifyErrorEvent, VerifyDoneEvent

int32 CreateImage (SCreateImageParams cCreateImageParams, int8 nTaskType);

**Parameters:**

SCreateImageParams cCreateImageParams

int8 nTaskType

**Special Data Type:**

**struct ScreateImageParams**

{

TCHAR lpszImagePath[_MAX_PATH];

TCHAR lpszBadSectorsFilePath[_MAX_PATH];

int16 nImageType;

SReadErrorCorrectionParams cErrorParams;

}

**TCHAR lpszImagePath[_MAX_PATH]**
The path to save image to

**TCHAR lpszBadSectorsFilePath[_MAX_PATH]**
The path to save a file with a list of bad sectors. This field can be empty.

**int16 nImageType**
Type of image you want to create. Possible options: BS_IMG_ISO and BS_IMG_BIN. Use GetPossibleImageFormats to get possible formats.

**SReadErrorCorrectionParams cErrorParams**
Read error correction settings

**struct SreadErrorCorrectionParams**

```
{
        BS_BOOL bErrorCorrection;
        BS_BOOL bBlankBadSectors;
        uint8 nHardwareRetryCount;
        uint8 nSoftwareRetryCount;
}
```

**BS_BOOL bErrorCorrection**
BS_TRUE - Error correction is enabled, BS_FALSE - Error correction is disabled. Sectors with read errors are skipped.

**BS_BOOL bBlankBadSectors**
BS_TRUE - in case of reading errors, the sector will be filled with zeros, BS_FALSE - sector containing error will be written "as is".

**uint8 nHardwareRetryCount**
The number of hardware read retries

**uint8 nSoftwareRetryCount**
The number of software read retries

### *GetPossibleImageFormats()*

Allows you to get a list of available formats for the image create. pImageFormats - return value. It is a combination of flags BS_IMG_ISO and BS_IMG_BIN.

int32 GetPossibleImageFormats (int16 *pImageFormats);

**Parameters:**

**int16 *pImageFormat**
A combination of flags BS_IMG_ISO and BS_IMG_BIN

## Disk/Image Edit

The FoxBurner SDK contain a feature to read from disks and disksimages, similar to tools like PowerISO, MagicISO or ISOBuster. This functionality allows you to make clones of those software but also a simple BluRay Reader for WindowsXP.

The functionality is shown in the sample “ImageEditFox”

### *SetReadDevice()*

With thisfunction you will set the device where the SDK will read the disk from. For diskImages you need to set the Imagewriter as default device and or select a own devicename with the first char “>” like >: test1.iso. For Diskimages you need to call further SetImageFilePath().

### *GetMediumInformation()*

Read information about specified medium.

## *GetTrackInformation()*

Read information about specified tracks.

## *GetSessionInformation()*

Read information about specified session.

## *OpenDiskSession()*

Open disk session for further operations.

int32 OpenDiskSession (int32 index, int32 nTrackNumber, HSESSION *phSession, int32 nFileSystem = BS_FS_UNKNOWN);

**Parameters:**

**int32 index**
Device index

**nTrackNumber**
Track number to read file system

**phSession**
Pointer to return opened session handle

**nFileSystem**
File system type to open´

## *GetUDFVolumeInformation()*

Returns UDF volume information from opened session. Session should be opened with nFileSystem == BS_FS_UDF.

int32 GetUDFVolumeInformation (HSESSION hSession, struct SUDFVolumeInfo* pUDFVolumeInfo);

**Parameters:**

**HSESSION hSession**
Handle to an open session.

**struct SUDFVolumeInfo* pUDFVolumeInfo**
UDF volume information structure

**Special data type**

```
struct SUDFVolumeInfo
{
        TCHAR chVolumeLabel[128];
        TCHAR chPreparer[128];
        int32 nVersion;
        int32 nPartitionType;
        int32 nFileCount;
        int32 nDirCount;
};
```

**TCHAR chVolumeLabel[128];**
Volume label.

**TCHAR chPreparer[128];**
Identifier of the application that prepared the disk.

**int32 nVersion;**
UDF verision. Possible values: BS_UDF_VERSION_(102, 150, 200, 201, 250, 260)

**int32 nPartitionType;**
UDF partition type. Possible values: BS_UDF_PARTITION_(PHYSICAL, VIRTUAL, SPARABLE)

**int32 nFileCount;**
Number of files in the volume.

**int32 nDirCount;**
Number of directories on the volume.

## *GetISOVolumeInformation()*

Returns ISO/Joliet volume information from opened session. Session should be opened with nFileSystem == BS_FS_ISO9660 or nFileSystem == BS_FS_JOLIET.

int32 GetISOVolumeInformation (HSESSION hSession, struct SISOVolumeInfo* pISOVolumeInfo);

**Parameters:**

**HSESSION hSession**
Handle to an open session.

**struct SISOVolumeInfo* pISOVolumeInfo**
ISO 9660/Joliet volume information structure.

**Special data type**

```
struct SISOVolumeInfo
{
        TCHAR chVolumeLabel[128];
        int32 nVolumeDescriptorAddress;
        int32 nVolumeSize;
        int32 nRootAddress;
        int32 nPathTableAddress;
        int32 nPathTableSize;
        struct SISOInfoEx sInfoEx;
};
```

**TCHAR chVolumeLabel[128];**
Volume label.

**int32 nVolumeDescriptorAddress;**
Volume descriptor address (LBA).

**int32 nVolumeSize**;
Volume size in sectors.

**int32 nRootAddress;**
Root directory descriptor location (LBA).

**int32 nPathTableAddress;**
Location of path table (LBA).

**int32 nPathTableSize;**
Size of path table in bytes.

**struct SISOInfoEx sInfoEx;**
Extended ISO information.

## *GetBootVolumeInformation()*

Returns UDF volume information from opened session. Session should be opened with nFileSystem == BS_FS_BOOTABLE.

int32 GetBootVolumeInformation (HSESSION hSession, struct SBootVolumeInfo* pBootVolumeInfo);

**Parameters:**

**HSESSION hSession**
Handle to an open session.

**struct SBootVolumeInfo* pBootVolumeInfo**
Boot volume information structure.

**Special data type**

```
struct SBootVolumeInfo
{
        int32 nVolumeDescriptorAddress;
        struct SBootInfoEx sInfoEx;
};
```

**int32 nVolumeDescriptorAddress;**
Boot volume descriptor location (LBA).

**struct SBootInfoEx sInfoEx;**
Extended boot information.

## *ReadFileContents()*

Read nBufferSize bytes from file lpszFilePath from session hSession.

int32 ReadFileContents (HSESSION hSession, const TCHAR* lpszFilePath, int64 nOffset, void* pBuffer, int32 nBufferSize, int32* pRead);

**Parameters:**

**HSESSION hSession**
Handle of opened session

**const TCHAR* lpszFilePath**
path to the file inside session

**int64 nOffset**
Byte offset in file

**void* pBuffer**
Pointer to the output buffer

**int32 nBufferSize**
Size of output buffer

**int32* pRead**
Returns number of bytes readed

### *SaveTrackToFile()*

Saves track from the disk to file.

int32 SaveTrackToFile (int16 nTrackNumber, const TCHAR* pchFileName, int32 nFileFormat);

**Parameters:**

**int16 nTrackNumber**
Track to save

**const TCHAR* pchFileName**
File path to save track to

**int32 nFileFormat**
Output file format.

### *ReadSectors()*

Read sectors from medium to a buffer.

int32 ReadSectors (int32 nLBA, int32 nCount, int8 nFormat, void* pBuff, int32 nBuffLength);

**Parameters:**

**int32 nLBA**
First sector location

**int32 nCount**
Number of sectors to read

**int8 nFormat**
Read format Possible values are: BS_IMG_ISO - read 2048 sectors and BS_IMG_BIN - RAW read - 2352 sectors

**void* pBuff**
Pointer to the output buffer

**int32 nBuffLength**
Length of the buffer

## CD Text Read

CD Text function allow you to read the CD Text Information out of a inserted Audio CD / DVD. You can read the fields:

| Field Name | **Field ID** |
|---|---|
| BS_CDTCI_ARRANGER | Arranger |
| BS_CDTCI_COMPOSER | Composer |
| BS_CDTCI_TITLE | Track/Disc title |
| BS_CDTCI_MESSAGE | Message Text |
| BS_CDTCI_PERFORMER | Performer |
| BS_CDTCI_SONG_WRITER | Song Writer |

### *GetCDTextHandle()*

This function create a handle to a Audio CD /DVD.

int32 BS_CALL  GetCDTextHandle(int32 index, int32 *pHandle);

**Parameters:**

**int32 index**
The index to the drive that contains the Audio Disk.

**int32 *pHandle**
A pointer to a handle that will contain the CD Text handle.

### *GetCDTextItem()*

This function will receive the CD Text information of a given field inside the CD Text structure. pchItemText first should be passed equal to 0, to learn the required buffer length, stored in pnLen.

int32 BS_CALL  GetCDTextItem(int32 hOpenedHandle, int32 nTrackNumber, int32 nCDTCI, TCHAR* pchItemText, int32 *pnLength);

**Parameters:**

**int32 hOpenedHandle**
Handle that was created with GetCDTextHandle()

**int32 nTrackNumber**
The number of the Track you want to get information from.

**int32 nCDTCI**
The identifier of the field you want information from.

**TCHAR* pchItemText**
A Pointer to a text buffer that will contain the information.

**int32 *pnLength**
The length of the text information.

### *CloseCDTextHandle()*

This function will close a created handle.

int32 BS_CALL  CloseCDTextHandle(int32 hOpenedHandle);

**Parameters:**

**int32 hOpenedHandle**
Handle that was created with GetCDTextHandle()

## CD Text Write

With AudioCD and MixedModeCD Projects you can either write CDText. The FoxBurner SDK supports Unicode and Multibyte.
Additional Values are: MCN, ISRC and indexes

### *SetAudioFileProperty()*

With this function you can set the CDText for CD and Track.

int32 BS_CALL  SetAudioFileProperty(struct SFileAudioProperty FileToAdd);

**Parameters:**

**const TCHAR* lpszSourceFilePath**

Disc CDText
track number or path to audio root. Audio project = Set lpszSourceFilePath to "\\". MixedMode = Set lpszSourceFilePath to "\\audio

Track CDText
Set lpszSourceFilePath to "NN", where NN is a track number. For example: "01", "02", "10", "11". Tracks are numbered from 1.

**Structure SfileAudioProperty:**

SAudioGrabbingParams structure defines parameters for encoding and tagging

struct SFileAudioProperty

{

**const TCHAR* lpszTitle**
Track/Disk Title

**const TCHAR* Performer**
Track/Disk Performer

**const TCHAR* SongWriter**
Track/Disk Songwriter

**const TCHAR* Composer**
Track/Disk Composer

**const TCHAR* Arranger**
Track/Disk Arranger

**const TCHAR* Message**
Track/Disk Message

**const TCHAR* lpszMCN_ISRC**
MCN or ISRC depending on lpszSourceFilePath

**int nPause**
Pause before track in seconds

**const int32* pIndexes**
Array of indexes inside track. can be NULL

**int32 nIndexesLength**
Number of items in pIndexes

}

## *SetWriteCDTextInUnicode()*

Set CD-Text encoding. If bWriteInUnicode is BS_TRUE, then CD-Text will be encoded in UCS-2, else CD-Text will be encoded in ASCII char set. In the second case, all non-valid symbols will be converted to underscores.

int32 BS_CALL SetWriteCDTextInUnicode (BS_BOOL bWriteInUnicode = BS_FALSE);

**Parameters:**

**BS_BOOL bWriteInUnicode**
True will write Unicode, False will write Multibyte. Default is Multibyte.

## *GetWriteCDTextInUnicode()*

Get current CD-Text encoding.

int32 BS_CALL GetWriteCDTextInUnicode (BS_BOOL* pbWriteInUnicode);

**Parameters:**

BS_BOOL* pbWriteInUnicode
A pointer to a BOOL value that will receive the current state.

# AudioGrabbing

The FoxBurner 5 and above allow you to grab Audio files out of Audio CD Content. Just get the Track comtent of the AudioCD and call the GrabFunction.
The AudioGrabbing functions you will find inside the Sample “ImageEdit”.

## *SetReadDevice()*

With thisfunction you will set the device where the SDK will read the disk from. For diskImages you need to set the Imagewriter as default device and or select a own devicename with the first char “>” like >: test1.iso. For Diskimages you need to call further SetImageFilePath().

## *GetMediumInformation()*

Read information about specified medium.

## *GrabAudioTrack()*

Audio disk grabbing function

int32 BS_CALL  GrabAudioTrack (SAudioGrabbingParams cAudioGrabbingParams, int16 nTrackNumber, const TCHAR *strSavePath);

**Parameters:**

**SAudioGrabbingParams cAudioGrabbingParams**
Object defines grabbing and tagging parameters

**int16 nTrackNumber**
Number of track on AudioCD disk

**const TCHAR *strSavePath**
Full path to save encoded track

**Structure SaudioGrabbingParams:**

struct SaudioGrabbingParams

{

**int32 nBitrate**
Field for value of constant bitrate

**int32 nMinBitrate**
Field for minimal value of bitrate for variable bitrate

**int32 nMaxBitrate**
Field for maximal value of bitrate for variable bitrate

**uint8 nBitrateType**
Field for bitrate type

**BS_BT_VARIABLE**
Encode with variable bitrate

**BS_BT_CONSTANT**
Encode with constant bitrate

**uint8 nEncoderType**
Field for encoder type

**BS_ET_MP3**
Mp3 encoder

**BS_ET_WMA**
WMA encoder

**BS_ET_AAC**
AAC encoder

**BS_ET_OGG**
Ogg encoder

**BS_ET_MP4**
Mp4 encoder

**uint8 nTagChoice**
Field for tag choice

**BS_TCH_NONE**
Don't need add tags

**BS_TCH_CDTEXT**
Add tags from CD-TEXT only

**BS_TCH_INTERNETDB**
Add tags from Internet data base only

**BS_TCH_CDTEXT_INTERNETDB**
Add tags from CD-TEXT at first and than add missing tags from Internet data base

**BS_TCH_INTERNETDB_CDTEXT**
Add tags from Internet data base at first and than add missing tags from CD-TEXT

**int32 nNetworkTagsHandle**
Field for handle of internet data base tags container

}

# Windows DRM

## *Working with DRM*

There are two types of DRM protections of Windows Media files: DMRv1 and DRMv7. In case of DRMv7 there are 2 ways of obtaining a license: silent and non-silent. Silent implies that a library (WMF SDK) will try to get a license automatically without user intervention. Non-silent – a user should open a page in a browser which will give him the ability to get a license manually.

In case of DRMv1 the only possible way is non-silent.

In case of DMRv7 non-silent acquisition the library monitors the process of the license acquisition and in case of success reopens the given DRM protected file. To stop the monitoring it is necessary to call the method StopLicenseAcquisition.

In case of DRVv7 silent acquisition the library automatically reopens the file in case of successful acquisition of a license.

In case of DRMv1 it is necessary to manually reopen a file after the license has been acquired.

## *Functions*

GetDRMEventCallback

SetDRMEventCallback

GetDRMLicenseEventCallback

SetDRMLicenseEventCallback

These functions give the ability to get or set callback routines which will be called by DRM subsystem. If a user doesn't provide callback routines then the following error will be returned during the attempt to add DRM protected file:

BS_SDK_ERROR_INVALID_FILE_FORMAT.

StopLicenseAcquisition

This method stops the license acquisition process. (see below)

## *DRM Event*

DRM Event. It is called when an event connected to DRM occurs and requires user input. Prototype:

__int32 (*DRMEvent)(const TCHAR* pcFileName, __int32 nEventCode, void *pUserData);

**pcFileName**
name of DRM protected file which caused the event

**nEventCode**
See event code

**pUserData**
Data provided by the user

**Event codes:**

**BS_DRM_EVENT_UNTRUSTED_URL**
This event is signaled during the attempt to open a DRM protected file when the license URL is not trusted. The callback should return BS_TRUE to continue to open the process or BS_FALSE to interrupt it.

**BS_DRM_EVENT_TAMPERED_URL**
This event is signaled during the attempt to open a DRM protected file when the license URL is tampered. The callback should return BS_TRUE to continue to open the process or BS_FALSE to interrupt it.

**BS_DRM_EVENT_CURRUPTED_URL**
This event is signaled during the attempt to open a DRM protected file when the license URL signature is corrupted. The callback should return BS_TRUE to continue to open the process or BS_FALSE to interrupt it.

**BS_DRM_EVENT_ACQUISITION_TYPE**
This event is signaled during the attempt to open a file protected with DRMv7 to give the user the possibility to choose a type of license acquisition. The callback should return:

1. BS_DRM_LICENSE_NONSILENT – non-silent license acquisition
2. BS_DRM_LICENSE_SILENT - silent license acquisition
3. BS_DRM_LICENSE_ABORT – do not do anything

**BS_DRM_EVENT_LICENSE_ACQUIRED**
This event is signaled when the license has been successfully obtained.

**BS_DRM_EVENT_LICENSE_NOT_ACQUIRED**
This event is signaled when it was impossible to obtain a license.

**BS_DRM_EVENT_NEED_INDIVIDUALIZATION**
This event is signaled when the OS requires a security upgrade. If the user agrees to download additional components from the Internet then the callback should return BS_TRUE, otherwise – BS_FALSE.

## *DRM License Event*

DRM License Event. It is called during non-silent license acquisition to pass the user the license URL and license settings.

void (*DRMLicenseEvent) (const TCHAR* pcFileName, struct SDRMLicenseInfo LicenseInfo, void* pUserData);

**pcFileName**
Name of DRM protected file that caused the event

**LicenseInfo**
See license information

**pUserData**
Data provided by the user

**Structure SDRMLicenseInfo:**

```
struct SDRMLicenseInfo
{
    const WCHAR* szLicenseURL;
    const char* szPostData;
};
```

**szLicenseURL**
URL which gives the possibility to get a license.

**szPostData**
Additional data to be passed using a POST request. This value may be equal to zero in case of protection with DRMv1. The example of using the structure and working with a browser can be found in the AudioSample::AudioSampleDlg.cpp.