Qflix Pro API

The “Qflix Pro API” is an extension of Sonic’s existing AuthorScript Storage Device (AS_StorageDevice) SDK. AS_StorageDevice exposes device enumeration, device properties, media properties, volume mounting, and some volume writing. The new items specific to Qflix Pro are Device and Media properties regarding CopyProtection, as well as new flags when writing Image files.

Overview

This document describes how to write a normal 2048-byte-per-block disc image from the harddrive to a CSS-protected disc. The caller needs to know the full path and name of image file to write, plus 2 small binary files from the DVD-CCA (a Disc Key Block and a Title Key Block). CSS is added at write time. No temporary files are created in this process.

DVD-CCA provided files

The DVD-CCA may provide several binary files as a bundle of CSS keys. The file named SMA3_ED.BIN is a Disc Key Block. The file named SMA3_ET.BIN is a Title Key Block. Any other files provided as a key bundle are not used for Qflix.

Example Code

This document should accompany example code “QflixSample.cpp”. The example code can be copied or step through for demonstration.

Base Headers Required

Using AS_StorageDevice will require the AS_StorageDevice library and a few headers for definitions. The shortest list:

```
#include "AS_StorageDevice.h"
#include "AS_StorageTypes.h"
#include "AS_StorageError.h"
#include "AS_StringHelper.h"
```

Detecting Devices

Create an AS_StorageDevice object and scan for devices. Later, iterate through the device list to determine device properties on each one.

```
AS_StorageDevice device;
UInt32 OpticalCount = 0;
retVal = AS_GetStorageDeviceCount(AS_StorageDevice::Scan_Optical |
AS_StorageDevice::Scan_Tape, OpticalCount);
```

Opening Devices

A Device must be opened before using it, and closed when the caller is finished. The devices are numbered 1..N, where N is the OpticalCount received in the call to AS_GetStorageDeviceCount.

```
retVal = AS_OpenStorageDevice(1, device);
```

Obtaining Exclusive Access

Recommended; applications may like to gain exclusive access to the device and media before lengthy operations and before changing the media contents.

```
AS_StringA currentUsedApp(AppName);
retVal = AS_StorageDevice_ExclusiveAccess(device,
AS_StorageDevice::ExclusiveAccess_Obtain, AS_StringA("AS_StorageTest"), currentUsedApp);
if (retVal != AS_Error_None)
{
        OutputError(retVal);
        return retVal;
}
```

Device Properties

Once a StorageDevice is open, we can query the device properties. These properties do not require a media to be in the drive. A full list of AS_StorageDevice::DevProp_Type can be found in "AS_StorageDevice.h". Specifically for Qflix Pro, the property to query shall be DevProp_Type::DevProp_CopyProtection.

```
ProtectionType PropVal=0;
UInt32 ret_data_size=0;
err = AS_StorageDevice_GetDeviceProperty(device,AS_StorageDevice::DevProp_CopyProtection,
sizeof(PropVal), &PropVal, &ret_data_size);
```

In the case of DevProp_CopyProtection, a bitflag is returned, indicating what types of Copy Protection this device supports.

```
// returned during query of DevProp_CopyProtection and MedProp_CopyProtection
typedef UInt32 ProtectionType;

// drive/media offers no copy protection
static const ProtectionType ProtectionType_None             = 0;

// drive/media offers Qflix Pro v1.0 CSS protection.
static const ProtectionType ProtectionType_Qflix_CSS        = (1 << 0);

// drive/media offers GuardBlock "Bad ECC blocks"
static const ProtectionType ProtectionType_GuardBlock_BadECC = (1 << 16);
```

Checking Media State

Caller may check media properties after media is inserted. To check for media, use AS_StorageDevice_GetState.

```
enum DeviceState    // device state
{
        State_Not_Ready       = 70,   // generally no media in device
        State_Ready           = 17,   // device is ready
        State_Becoming_Ready  = 71    // device is busy
};
AS_Error err;
AS_StorageDevice::DeviceState State;
AS_StorageDevice_GetState(device, &State);
```

Media Properties

If a media is inserted, caller may inspect properties of the media. A full list of AS_StorageDevice::MediaProp_Type can be found in "AS_StorageDevice.h". Specifically for Qflix Pro, the property to query shall be

MediaProp_Type::MedProp_CopyProtection. The return value is the same enum as in DevProp_CopyProtection.

```
ProtectionType protection;
err = AS_StorageDevice_GetMediaProperty(device,
AS_StorageDevice::MedProp_CopyProtection,sizeof(protection), &protection,
&ret_data_size);
```

Writing a Raw Image to Qflix Media

The simplest Qflix Pro writing example to make is the case of a Raw Image File on the harddrive. AS_Storage is not required to modify this type of image – it is "ready to burn". If the caller has already determined the drive and media support Qflix Pro writing, we are ready to begin. AS_Storage SDK adds CSS in realtime while writing. We can write this image file to a normal drive or Qflix Pro drive with the call:

```
AS_StorageError AS_API AS_StorageDevice_Copy (
const AS_StorageDevice &           srcDevice, // IN: source device
AS_StorageDevice::FileFormatType    format,  // IN: format type setting for dest file(s)
AS_StorageDevice::CopyFlags           flag,  // IN: flags for copy params
UInt32         trackNum,      // IN: optional param to copy specific track from source
UInt32         NumDevices,    // IN: num of destinations
const AS_StorageDevice * DeviceList,         // IN: array of destination devices
AS_StorageDevice::InfoCallback callbackList, // IN: progress callback function
void* callerUse);                            // IN: progress callback userdata
```

Usually when using the AS_StorageDevice_Copy() function, the source device (srcDevice) should have it's `devicepath` member set to the path of the image being written, and its type set to Type_File.

```
srcDevice.deviceType = AS_StorageDevice::Type_File;
srcDevice.devicePath = AS_StringA(imagefilepath);
```

When using an XML file to describe the copy job, however, this srcDevice parameter may be an empty AS_StorageDevice. You do not need to set up parameters since the XML file completely describes the copy job.

The destination device (destDeviceList) should have the `deviceXMLOptionsPath` member set to the XML file containing the Qflix options XML file.

```
device.deviceXMLOptionsPath = AS_StringA(XMLFilePath);
```

The Qflix Pro CSS encryption will be done while it writes if we use the CopyFlag set to CopyFlags_MD. If we set the CopyFlag to CopyFlags_ISO, then the disc is produced without encryption.

Similarly, another flag can be used to write an Image File using Sonic "CopyBlock" protection.

A complete set of calls can be copied and pasted from the example code "QflixSample.cpp" accompanying this documentation.

Optional : AES Encryption of Source Image and Source Keys
The SDK user may have reason to obfuscate their CSS keys, their source ISO images, or both. These items can be encrypted using a command-line tool provided with the SDK : *AesEncrypt.exe*. The encryption this tool adds is AES in "CNTR" mode. An introduction to AES can be found on the web:
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

In the case of CSS keys – either the Disc Key Block or the Title Key Block – the AesEncrypt tool will output only one file; a package containing both the AES key and the encrypted content (caller's CSS keys). The AES key in the package is encrypted using another encryption standard called "ECC". The CSS keys are only of use to the SDK when it writes a CSS disc. No tool is given to open this package – only the SDK knows how to open it.

In the case of ISO disc images, AesEncrypt parses the filesystem to learn where the video content is. Much like a finished CSS disc, it only encrypts the video content (VOBs). The output from the AesEncrypt tool is two files – a key vault and a newly encrypted ISO image. These two items must be presented together to the SDK for it to undo the AES encryption on the disc image.

AesEncrypt usage examples
To encrypt a CSS Disc Key Block or Title Key Block with AES:
```
AesEncrypt.exe /FILE <source> /OUT <destination> /TYPE FILE /CUSTOMER nnnn
```
Example:
```
AesEncrypt.exe /FILE c:\unencryptedDKB.dat /OUT c:\encryptedDKB.dat /TYPE FILE /CUSTOMER
1234
```

In this example, the output file is the vault "encryptedDKB.dat". You may use any customer number 0 through 9999; it acts as the public key to open your vault. Only the SDK knows the private key. You will need to pass the customer number you used as an XML parameter for the SDK to open this vault. (example XML is shown later).

To encrypt an ISO image (video data only) with AES:
```
AesEncrypt.exe /FILE <source> /OUT <destination> /KEY <output key name> /TYPE ISO
/CUSTOMER nnnn
```
Example:
```
AesEncrypt.exe /FILE c:\unencryptedSource.iso /OUT c:\encryptedSource.iso /KEY aeskey.dat
/TYPE ISO /CUSTOMER 1234
```

In this example, the output ISO file with video data encrypted is "encryptedSource.iso". The output AES key vault associated with this ISO image is "aeskey.dat". The SDK needs the ISO, the vault, and your chosen customer number to write this ISO image later.

The section titled "AS_StorageDevice XML options" describes how to pass the AES Encrypted keys or Image files to the SDK.

AS_StorageDevice XML options
The XML file describing the copy job is expected to have several parameters. Example XML files are in this section.

The XML should begin and end with a node called <job_parameters> .

It should have nodes called <source> and <output>.

Source
A source is an "ISO Image", or "raw image file". The Qflix API expects to receive the path to a raw image in 2048-bytes-per-sector mode. It belongs in the <ISO> section. The image would have a <source_path> and <format_flags>. Possible only flag enabled now for format_flags is : normal. All other flags have become obsolete.

If the source material image file is AES encrypted, then there is an additional parameter <source_AES_key_path>. This has one parameter named "index" which is the customer number (public key) needed to open the AES key vault. An encrypted example is shown in the "Example XML Files" section.

Output
The <output> node describes the disc that will be produced.

Output shall have a <burn_method> node to specify which type of disc is being produced. Valid flags are : production | simulate_protection | simulate_write | simulate_failure. Example: "production" flag produces the CSS disc on Qflix Pro hardware. "simulate_protection" will produce a non-encrypted disc on normal hardware. "simulate_write" may not use any media at all. "simulate_failure" will cause an error to be reported and no media consumed.

The <replicator_CSS> node describes the CSS-specific files needed to complete the disc. The CSS-Specific files are <disc_key_block_path> and <title_key_block_path>. If the CSS files are AES encrypted, then the tags should instead be named <encrypted_disc_key_block_path> and <encrypted_title_key_block_path>. Both of these have a parameter named "index" which serves as the customer number (private key) needed to open the AES vault. An encrypted example is shown in the "Example XML Files" section.

Optionally we can turn on "CopyBlock" protection with the <copyblock> node. There are no special parameters in this node.

Disc_Key_Block_File
The full path to the Disc Key Block File. This file shall be 2048 bytes. The DVD-CCA provides this binary file. If you do not yet have a CSS license, you may use a dummy file of the correct size and the simulate_protection flag for testing.

Title_Key_Block_File
The full path to the Title Key Block File. This file shall be 495 bytes (99 titles times 5 bytes each). The DVD-CCA provides this binary file. If you do not yet have a CSS license, you may use a dummy file of the correct size and the simulate_protection flag for testing.

Encrypted Disc Key Block File
The full path to the AES-encrypted Disc Key Block file.

Encrypted Title Key Block File
The full path to the AES-encrypted Title Key Block file.

Example XML files

An example of a complete XML file describing an AS_StorageDevice_Copy job is shown here. This example did not use AES encryption on the ISO image or the CSS keys. The items in red color are changeable on a per-job basis.

```
<?xml version="1.0" encoding="UTF-8"?>
<job_parameters>
	<version>1.0</version>
	<source>
		<ISO>
			<source_path>c:\_uploads\sonic.iso</source_path>
			<format_flags>normal</format_flags>
		</ISO>
	</source>
	<output>
		<burn_method>production</burn_method>
		<replicator_CSS>
			<disc_key_block_path>c:\_uploads\dkb.dat</disc_key_block_path>
			<title_key_block_path>c:\_uploads\tkb.dat</title_key_block_path>
		</replicator_CSS>

	<copyblock>
	</copyblock>

	</output>
	<comment>You can stick anything in here</comment>
</job_parameters>
```

An example of a complete XML file describing an AS_StorageDevice_Copy job with AES encryption is shown here. Both the source ISO image and the CSS keys are encrypted. Note that “index” is the customer number that you chose to use when using the AesEncrypt tool. The items in red color are changeable on a per-job basis.

```
<?xml version="1.0" encoding="UTF-8"?>
<job_parameters>
	<version>1.0</version>
	<source>
		<source_AES_key_path index="0"> c:\_uploads\SONIC_AESCNTR.ISO.KEY
		</source_AES_key_path>
		<ISO>
			<source_path>c:\_uploads\SONIC_AESCNTR.ISO</source_path>
			<format_flags>normal</format_flags>
		</ISO>
	</source>
	<output>
		<burn_method>production</burn_method>
		<replicator_CSS>
			<encrypted_disc_key_block_path index="0">
			c:\_uploads\dkbaescntr.dat
			</encrypted_disc_key_block_path>

			<encrypted_title_key_block_path index="0">
			c:\_uploads\tkbaescntr.dat
			</encrypted_title_key_block_path>
		</replicator_CSS>

	</output>
	<comment>You can stick anything in here</comment>
</job_parameters>
```