Thunderbolt™

Software Development Kit Guide

Document Version 1.12

Document Release Date: August, 2020

License

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Legal Notices and Disclaimers

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A "Mission Critical Application" is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS' FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order. Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel, Thunderbolt, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

Table of Contents

Version History

1.12	August 16, 2020	- Add DeviceImageContainerUtilities - Add RetimerImageUtilities - Utilities.RetrieveDeviceIdFromImage - Utilities.GetDeviceType

1.11	July 8, 2020	- Added support for GR FW update through USB3. - Added support for GR on board retimer FW update. - Added useful fields to DeviceParams class. - Added useful fields and methods to IDriverDevice interface. - Added ISettings to allow sample code to override some of DriverApi functionality. - Added FwUpdateProgressFactory.NumOfImagesToUpdate, so sample code can control how many images in one process will be updated. Useful for updating through container. - Added Drom and related structures, to work with DROM effectively. Works on USB4 devices and retimers. - Added retimer related structures: RetimerRoute, IOnBoardRetimer interface, IDriverDevice.OnBoardRetimers. - Added a few new exceptions. **GR FW update through USB3 high level flow:** DriverFactory.GetDevices() returns a list of devices, those connected through USB3 will have the field – DeviceParams. ConnectedThroughUsb set. These devices are shown in the FwUpdateTool without being under Host controller hierarchy. The image validation and FW update process APIs weren't change for these devices. The flow is encapsulated in the FwUpdateDriverApi. **DBR FW update high level flow:** The flow starts with the call for DriverFactory.GetDevices(). Each device may have OnBoardRetimers list not empty. Once user chooses a device in one of the sample tools and starts FW update, the sample searches for the proper DBR image on the chosen device. It validates the image using Intel USB4 validator + IOnBoardRetimer.Drom and uses IOnBoardRetimer.UpdateFirmware to update DBR firmware. **GR Container update high level flow:** The container might be utilized to support one binary image, which includes the GR and DBR images. The SDK splits the container and performs FW update, for each image one by one starting from DBRs. Script to prepare the container can be found in the top level directory - *gr_container_generator.py*. Usage information can be retrieved by running it with -h flag. **USB4 image validation high level flow:** If a device is USB4/(DBR retimer) and Intel made, the validation uses DROM information in the device and in the image to validate the provided NVM against the image on a device before FW update. - Non Intel device NVM validation is not supported, as it is unknown where in the NVM the DROM can be found.

1.10	July 22, 2019	- Add new exception I2C_ACCESS_NOT_SUPPORTED, which is thrown when methods that access I2C are used on unsupported I2C devices.
1.9	May 30, 2019	- See 0x229 exception description update.
1.8	April 2, 2019	- Fix the issue with wrong NVM version format.
1.7	December 20, 2018	- Remove usage of CLIAdapter, which was written in unmanaged code. RCR-1305945253 - Improve status progress update in host to include the time wait for power cycles. - Added logger and option to inject it to the closed driver API. - Added support for ICL. - A new exception UNSUPPORTED_OPERATION, which is thrown when not allowed methods are used on ICL. - Updated documentation to include the public convenience methods in Utilities. - Provided Exceptions class – FuDrvApiException from the FwUpdateDriverApi. - Updated the Samples to use the exceptions thrown from FwUpdateDriverApi.
1.6	October 16, 2018	- Retarget all the DLLs and samples to .Net 4.5.2. Due to request from HP.
1.5	September 30, 2018	- Change projects structure, to provide the legacy API as a sample. Legacy API - FwUpdateApiSample. - Added a new Driver API (FwUpdateDriverApi), used as the interface with a driver.
1.4	September 25, 2018	- Fix to support the changed driver API version.
1.3	August 8, 2018	- Add capability to identify if more than 1 handle open during FW update. - Added 2 new exceptions (0x22A, 0x22B). - All SDK components will be marked with the same version equal to the SDK version. - Fixed timeout issues during FW update. - Added appendix chapter with a list of the SDK's libraries details.
1.2	July 22, 2018	- Limitations from version 1.1 are solved in this version. - Added 2 new exceptions (0x228-0x229) - Update prerequisite chapter (added .Net and Redistributable)
1.1	Jule 8, 2018	- SDK API is functional, it supports all the operations described in this document. - The samples, where changed to use the non deprecated methods. - Added three new errors: 0x225, 0x226, 0x227.
1.0	June 17, 2018	- API is provided in the dll format. - The delivery must be used for OEM tools build enablement only. The API SDK DLL is not functional in this release.

1 Introduction

This guide describes how to use the Thunderbolt™ Software Development Kit (SDK).

The purpose of this guide is to provide information to OEMs that want to control the firmware update process of Thunderbolt controllers. This guide includes documentation only for the classes and methods that can be used to update the firmware of Thunderbolt controllers.

> **Note:**
>
> The SDK includes the API with the driver in dynamic library format (FwUpdateDriverApi.dll). It is located in the Sdk/ folder.
>
> The legacy API is implemented in a project called FwUpdateApiSample. It uses the FwUpdateDriverApi to communicate with the DCH driver. It is located in Samples/.
>
> For convenience and previous drops compatability, the legacy API is compiled as well and put in the Sdk/ folder. It's called FwUpdateApi.dll (as before)
>
> The SDK also includes three sample projects: DeviceFWUTool(CLI), FwUpdateTool (GUI) and FwUpdateCmd (CLI), located in the Samples folder. The samples show how to use the methods in FwUpdateApiSample to update the Thunderbolt firmware. The sample projects are written in C# using Microsoft* Visual Studio 2017. The API is written using .Net 4.5.2, so the sample applications are compiled using the same .Net version.

2 Prerequisites

These are the prerequisites to use this SDK:

- Supported operating systems:
  - Windows 10 64-bit RS4/RS5/19H1/19H2/20H1
- Supported Thunderbolt controllers:
  - L6000 Series and higher (both for host and for device)
- .Net 4.5.2 installed.

3 Thunderbolt FW Update package

This SDK includes some code to show how to implement the firmware update flow using the Thunderbolt API. The sample code is composed of an driver API dll, API sample and two sample applications (GUI based and CLI based) that use this API to perform firmware update. It can be used as reference when building other firmware update applications.

3.1 FwUpdateDriverApi

The API includes the interfaces with methods, required to communicate with the TBT DCH driver to perform firmware update related activities.

3.1.1 NvmVersion

Class to hold NVM version.

3.1.1.1 Major

```
public uint Major {get; set;}
```

NVM major version.

3.1.1.2 Minor

```
public uint Minor {get; set;}
```

NVM minor version.

3.1.2 ControllerParams

3.1.2.1 ControllerId

```
public string ControllerId {get;}
```

Retreive Controller Id as string. As it appears in device manager.

3.1.2.2 DeviceId

```
public ushort DeviceId {get;}
```

Retreive TBT device ID.

3.1.2.3 FullNvmVersion

```
public NvmVersion FullNvmVersion {get;}
```

Retreive full NVM version.

3.1.2.4 Generation

```
public uint Generation {get;}
```

Retreive TBT generation.

3.1.2.5 InSafeMode

```
public bool InSafeMode {get;}
```

Returns true if the controller is in safe mode, otherwise false.

3.1.2.6 OsNativePciEnum

```
public bool OsNativePciEnum {get;}
```

Returns true if the controller supports native pci enumeration, otherwise false.

3.1.2.7 Rtd3Capable

```
public bool Rtd3Capable {get;}
```

Returns true if the controller supports rtd3, otherwise false.

3.1.2.8 SecurityLevel

```
public uint SecurityLevel {get;}
```

Returns the controller's security level.

3.1.2.9 SupportsExternalGpu

```
public bool SupportsExternalGpu {get;}
```

Returns true if the controller supports external GPU, otherwise false.

3.1.3 DeviceParams

3.1.3.1 ControllerId

```
public strin ControllerId {get;}
```

Returns ID of the host controller the device is connected to.

3.1.3.2 ControllerNum

```
public byte ControllerNum {get;}
```

Returns TBT chip enumeration out of n chips in the device (as marked in the chip's NVM)

3.1.3.3 LinkSpeed

```
public byte LinkSpeed {get;}
```

Returns CIO link speed (in Gbps) of the device.

3.1.3.4 ModelId

```
public UInt16 ModelId {get;}
```

Returns Device model ID.

3.1.3.5 ModelName

```
public string ModelName {get;}
```

Returns Device’s model name.

3.1.3.6 VendorId

```
public UInt16 VendorId {get;}
```

Returns Device vendor ID.

3.1.3.7 VendorName

```
public string VendorName {get;}
```

Returns Device's vendor name.

3.1.3.8 NumOfControllers

```
public byte NumOfControllers {get;}
```

Returns number of TBT chips on the device.

3.1.3.9 NvmVersion

```
public NvmVersion NvmVersion {get;}
```

Returns NVM version of the device.

3.1.3.10 PortNum

```
public uint PortNum {get;}
```

Returns 0-based index of the port in the host controller the device is connected to.

3.1.3.11 PositionInChain

```
public uint PositionInChain {get;}
```

Returns 1-based index of the device position in the port the device is connected to.

3.1.3.12 Updatable

```
public bool Updatable {get;}
```

Returns true if the device is updatable, else false.

3.1.3.13 UUID

```
public string UUID {get;}
```

Returns device UUID – unique identifier ID.

3.1.3.14 ConnectedThroughUsb

```
public bool ConnectedThroughUsb { get; }
```

Returns if a device connected through USB3.

3.1.3.15 HasSharedNvmWithPd

```
public bool HasSharedNvmWithPd { get; }
```

Returns if a device shares NVM with PD controller.

3.1.3.16 IsUsb4Device

```
public bool IsUsb4Device { get; }
```

Returns if a device is USB4 device.

3.1.3.17 HasReadDromSupport

```
public bool HasReadDromSupport { get; }
```

Returns if a device supports read DROM.

3.1.4 IDriverBase

Base class for driver interface

3.1.4.1 GetNvmVersion

```
public NvmVersion GetNvmVersion()
```

Retreive NVM version from TBT entity.

3.1.4.2 I2CRead

```
public byte[] I2CRead(uint port, uint offset, uint length)
```

Read I2C register. May throw an exception with SDK_DRIVER_COMMUNICATION_ERROR code.

3.1.4.3 I2CWrite

```
public byte[] I2CWrite(uint port, uint offset, byte[] data)
```

Write I2C register. May throw an exception with SDK_DRIVER_COMMUNICATION_ERROR code.

3.1.4.4 ReadFirmware

```
public byte[] ReadFirmware(uint offset, uint length)
```

Read from NVM. May throw an exception with SDK_DRIVER_COMMUNICATION_ERROR code.

3.1.4.5 UpdateFirmware

```
public void UpdateFirmware(uint bufferSize, byte[] buffer)
```

Override current NVM with the given buffer content. May throw an exception with one of the following codes:

SDK_FW_UPDATE_MORE_THAN_ONE_HANDLE_OPEN, SDK_DRIVER_COMMUNICATION_ERROR,

SDK_ERROR_DURING_IMAGE_UPDATE, SDK_AUTHENTICATION_FAIL, SDK_FW_UPDATE_TIMEOUT

3.1.4.6 Logger

```
public ILogger Logger {get; set;}
```

Get or set the logger to use to log the activity.

3.1.5 IDriverController

Interface to controller

3.1.5.1 GetParams

```
public ControllerParams GetParams()
```

Returns the ControllerParams class - properties for this controller.

3.1.6 IDriverDevice

Interface to device

3.1.6.1 GetParams

```
public DeviceParams GetParams()
```

Returns the DeviceParams class - properties for this device.

3.1.6.2 OnBoardRetimers

```
IEnumerable<IOnBoardRetimer> OnBoardRetimers { get; }
```

An enumeration with OnBoardRetimers

3.1.6.3 ReadDrom

```
byte[] ReadDrom(uint offset, uint length);
```

Reads data from DROM (FW) of the device

3.1.6.4 Drom

```
IDrom Drom { get; }
```

Device's DROM

3.1.7 DriverFactory

Static class to get controllers or devices currently connected to the system.

3.1.7.1 GetControllers

```
public Dictionary<string, IDriverController> GetControllers()
```

Returns a dictionary, where each entry's key is controller ID and value is IDriverControler interface to connect to the controller.

```
public Dictionary<string, IDriverController> GetControllers(ILogger logger)
```

Same as above, but use logger to log the activity also sets the logger to all the controllers.

3.1.7.2 GetDevices

```
public Dictionary<string, IDriverDevice> GetDevices()
```

Returns a dictionary, where each entry's key is device ID and value is IDriverDevice interface to connect to the device. The dictionary includes an entry per connected Device. The supported connection types are TBT3, USB4, USB3.

```
public Dictionary<string, IDriverDevice> GetDevices(ILogger logger)
```

Same as above, but use logger to log the activity also sets the logger to all the devices.

3.1.7.3 Settings

```
public static ISettings Settings { get; set; } = new DefaultSettings();
```

Allows to get or set settings for FwUpdateDriverApi.

3.1.8 ILogger

Interface for a log implementation.

3.1.8.1 LogInfo

```
void LogInfo(string strMessage, [CallerMemberName] string member = "", [CallerFilePath] string
```

```
filepath = "", [CallerLineNumber] int line = 0)
```

Prints the informative log message into info channel.

3.1.8.2 LogErr

```
void LogInfo(string strMessage, [CallerMemberName] string member = "", [CallerFilePath] string
filepath = "", [CallerLineNumber] int line = 0)
```

Prints the error log message for error channel.

3.1.8.3 LogWarn

```
void LogInfo(string strMessage, [CallerMemberName] string member = "", [CallerFilePath] string
filepath = "", [CallerLineNumber] int line = 0)
```

Prints the warning log message for warning channel channel.

3.1.9 FuDrvApiException

Class to specify the exceptions which can be thrown by the FwUpdateDriverApi library.

3.1.9.1 ErrorCode

An enum that includes definition for the error codes as follows:

Value	Source	Description
0x225	DriverApi	Driver communication error
0x226	DriverApi	Error during image update
0x227	DriverApi	DCH driver is missing
0x228	DriverApi	The image provided is not signed. Therefore rejected.
0x229	DriverApi	A timeout has expired while FW update was performed. Not thrown during HR firmware update.
0x22A	DriverApi	More than one handle open during FW update.
0x22B	DriverApi	Driver API unknown.
0x22C	DriverApi	I2C access is not supported
0x22D	DriverApi	Adapter type is not supported
0x22E	DriverApi	Retimer sideband operation fail
0x230	DriverApi	Retimer Image format error

Value	Source	Description
0x231	DriverApi	Retimer access is not supported by the driver
0x232	DriverApi	Operation is not supported

3.1.10 FlashProgressEventArgs

Class to be used as a parameter by the ProgressUpdated event handler.

3.1.10.1 Progress

```
public uint Progress {get; set;}
```

Current status of firmware update progress, in percent.

3.1.11 IFwUpdateProgressWatcher

Interface to watch on progress update

3.1.11.1 ProgressUpdated

```
public event EventHandler<FlashProgressEventArgs> ProgressUpdated
```

Event, triggered each time the firmware update progress is updated.

3.1.12 FwUpdateProgressFactory

Class to get the flash progress interface

3.1.12.1 GetWatcher

```
public IFwUpdateProgressWatcher GetWatcher()
```

Returns an interface to watch on fw update progress.

3.1.12.2 NumOfImagesToUpdate

```
uint NumOfImagesToUpdate { get; set; }
```

This property allows to set or get the progress factor. NumOfImagesToUpdate is a number greater than 0. It is used by FW update progress watcher as progress denominator. It allows client to control progress update percentage for multiple images in one flow.

3.1.13 ISettings

Allows users to provide custom implementations to the following methods, which are being used in the FwUpdateDriverApi

3.1.13.1 HasSharedNvmWithPdController

```
bool HasSharedNvmWithPdController(ushort devId);
```

3.1.13.2 IsUsb4Device

Function which returns if the controller with a given device ID has shared NVM with PD.

```
bool IsUsb4Device(ushort devId);
```

3.1.13.3 HasOnBoardRetimers

Function which returns if the controller with a given device ID is USB4. This function is allows to override received from TBT Driver information regarding USB4 support by a device.

```
bool HasOnBoardRetimers(ushort devId);
```

3.1.13.4 GetDromAddressNvmLocation

```
uint GetDromAddressNvmLocation(ushort devId);
```

Function which returns the DROM location in the NVM of the controller with a given device ID. It is used in case device or driver doesn't support DROM read.

3.1.14 DromSectionName

Drom section name enum

```
Enum DromSectionName
{
   Header,
   VendorName,
   ModelName,
   ProductDescriptor = 9,
   SerialNumber,
   VendorSpecificOnBoardRetimers = 0x30
}
```

3.1.15 IDrom

3.1.15.1 Sections

```
Dictionary<DromSectionName, byte[]> Sections { get; }
```

Dictionary with each entry's key is drom section enum and the value is section in byte array.

3.1.15.2 Tbt3Compatible

```
bool Tbt3Compatible { get; }
```

Boolean indicating if DROM is TBT3 compatible.

3.1.16 DromFactory

3.1.16.1 CreateDrom

```
public static IDrom CreateDrom(Func<uint, uint, byte[]> dromReadFunc, ILogger logger)
```

This function allows to create IDROM given dromReadFunc.

3.1.17 Tbt3DromHeader

3.1.17.1 Constructor

```
public Tbt3DromHeader(byte[] rawHeader)
```

Constructs the TBT3 DROM header from byte array with the header content.

3.1.17.2 IsSame

```
public bool IsSame(Tbt3DromHeader other)
```

Returns if the Tbt3DromHeader is the same as provided other.

3.1.17.3 VendorId

```
public ushort VendorId { get; }
```

3.1.17.4 ModelId

```
public ushort ModelId{ get; }
```

3.1.17.5 ModelRevision

```
public byte ModelRevision { get; }
```

3.1.17.6 NvmRevision

```
public byte NvmRevision { get; }
```

3.1.18 Usb4DromProductDescriptor

3.1.18.1 Constructor

```
public Usb4DromProductDescriptor(byte[] rawProductDescriptor)
```

Constructs the USB4 DROM product descriptor from byte array with the proper content.

3.1.18.2 IsSame

```
public bool IsSame(Usb4DromProductDescriptor other)
```

Returns if the Usb4DromProductDescriptor is the same as provided other.

3.1.18.3 idVendor

```
public ushort idVendor { get; }
```

3.1.18.4 idProduct

```
public ushort idProduct { get; }
```

3.1.18.5 bcdDevice

```
public ushort bcdDevice { get; }
```

3.1.19 RetimerRoute

3.1.19.1 SmBusAddress

```
public ushort SmBusAddress { get; }
```

On board retimer address on SM BUS.

3.1.20 IOnBoardRetimer

3.1.20.1 Route

```
RetimerRoute Route { get; }
```

Retimer’s route, used to communicate with a retimer.

3.1.20.2 VendorName

```
string VendorName { get; }
```

Get the Device's vendor name

3.1.20.3 ModelName

```
string VendorName { get; }
```

Get the Device's model name

3.1.20.4 VendorId

```
string VendorName { get; }
```

Get the Device Vendor ID

3.1.20.5 ModelId

```
String VendorName { get; }
```

Get the Device Model ID

3.1.20.6 NvmVersion

```
NvmVersion NvmVersion { get; }
```

Get NVM version of the device

3.1.20.7 ReadNvm

```
byte[] ReadNvm(uint offset, uint length);
```

Reads data from retimer's NVM

3.1.20.8 ReadDrom

```
string VendorName { get; }
```

3.1.20.9 UpdateFirmware

```
string VendorName { get; }
```

3.1.20.10 Drom

```
string VendorName { get; }
```

3.2 FwUpdateApiSample

The API includes all the methods and fields as required by the sample applications. Additionaly it provides some utility methods for convenience. Only the main parts are documented here. For more information please refer to the source code and the additional documentation there.

Please note: All the functions throw exceptions in case of an error. Applications are responsible for catching the exception and displaying a reasonable error message to the user, if applied. CLI sample (see below) can be used as a reference for such handling.

3.2.1 SdkTbtBase

This class is a base class for the interfacing with both host controllers and devices.

This, with the derived classes (SdkTbtController and SdkTbtDevice), is the main interface of this API module for applications to use.

The main role of this class is to tie the interface to the driver but it also includes the interface for validating the compatibility of the new firmware image with the current controller.

3.2.1.1 UpdateFirmware

```
public void UpdateFirmware(UInt32 bufferSize, byte[] buffer)
```

A method to perform image update. This function throws an exception on error.

3.2.1.2 GetCurrentNvmVersion – Deprecated

Note: This method is deprecated and exists only for backward compatibility. Use GetCurrentFullNvmVersion method instead.

```
public UInt32 GetCurrentNvmVersion()
```

This method returns the current NVM version (major number only) by return value instead of output parameter. Throws an exception on error.

3.2.1.3 GetCurrentFullNvmVersion

```
public string GetCurrentFullNvmVersion()
```

This method returns the current NVM version (in major.minor format) . Throws an exception on error.

3.2.1.4 GetCurrentPdVersion – Deprecated

Note: This method is deprecated and exists only for backward compatibility. It is marked as obsolete and considered an error. Use I2CRead instead.

```
public string GetCurrentPdVersion()
```

This method returns "N/A".

3.2.1.5 I2CRead

```
public byte[] I2CRead(UInt32 port, UInt32 offset, UInt32 length)
```

This method reads the content of I2C register. Throws an exception on error.

3.2.1.6 I2CWrite

```
public void I2CWrite(UInt32 port, UInt32 offset, byte[]  data)
```

This method writes data to I2C register. Throws an exception on error.

3.2.1.7 ReadFirmware

```
public byte[] ReadFirmware(UInt32 offset, UInt32 length)
```

This method reads data from NVM (FW) of the controller/device. Throws an exception on error.

3.2.1.7.8 ValidateImage

```
public abstract void ValidateImage(string path)
```

This method validates that a given image (from the binary file in the given path) is valid as it relates to the existing image, existing hardware (controller), and the available area on the chip. This function uses a table of properties, which are compared between new image and existing one, and these match the properties that are described in the NVM release notes. We recommend comparing all these properties before updating the NVM image.

The method is implemented by the derived classes, SdkTbtController and SdkTbtDevice, and, in turn, these implementations use the ImageValidator class hierarchy.

3.2.1.8 UpdateFirmwareFromFile

```
public virtual void UpdateFirmware(string filename)
```

This method is a convenient utility for updating the firmware from a file, given the path to the file as an argument. It uses the UpdateFirmware method and throws an exception on error.

3.2.1.9 HasSharedNvmWithPd

```
public abstract bool HasSharedNvmWithPd { get; }
```

This method returns if a controller has shared NVM with PD.

3.2.2 SdkTbtController

This class is used to represent Thunderbolt host controller. Please note the differences in the methods interface as described in SdkTbtBase class documentation above.

It also includes the following static function for getting the available class instances.

3.2.2.1 GetControllersFromWmi - Deprecated

```
public static Dictionary<String,SdkTbtController> GetControllersFromWmi()
```

Note: This method is deprecated and exists only for backward compatibility. Use GetControllers() method instead.

This function enumerates all host controller instances that can be detected. All firmware update operations are performed on these instances.

3.2.2.2 GetControllers

```
public static Dictionary<String,SdkTbtController> GetControllers()
```

This function enumerates all host controller instances that can be detected. All firmware update operations are performed on these instances.

3.2.2.3 Fields

Type	Name	Value or Description
String	ControllerId	The ID of the host controller
UInt32	Generation	The generation of the host controller
UInt16	DeviceId	The ID of the host device
Boolean	IsInSafeMode	TRUE if the host controller is in safe mode
String	NvmRevision	NVM revision as read from the Drom, if is in Safe Mode or no drom returns NA
String	SecurityLevel	The host security level
Boolean	SupportsExternalGpu	TRUE if the host controller supports external GPU
Boolean	OsNativePciEnumeration	TRUE if the host controller is using OS native PCI Enumeration (Native Express mode) * Currently supported only on JHL6540/6340 controllers running FW NVM rev 21 and above
Boolean	RTD3Capable	TRUE if the host controller supports D3 power state
String	VendorId	Vendor Id, as it comes from the Host DROM section
String	ModelId	Model Id, as it comes from the Host DROM section
String	ModelRevision	Model Revision, as it comes from the Host DROM section
String	CustomizedTIVersion	Customized TI Version as read from the Drom, if is in Safe Mode or no drom returns NA

3.2.3 SdkTbtDevice

This class is used to represent a Thunderbolt device.

It also includes the following static function for getting the available class instances.

3.2.3.1 GetDevicesFromWmi - Deprecated

```
public static Dictionary<String,SdkTbtDevice> GetDevicesFromWmi()
```

Note: This method is deprecated and exists only for backward compatibility. Use GetDevices() method instead.

This function enumerates all device controller instances that can be detected. All firmware update operations are performed on these instances.

3.2.3.2 GetDevices

```
public static Dictionary<String,SdkTbtDevice> GetDevices()
```

This function enumerates all device controller instances that can be detected. All firmware update operations are performed on these instances.
In case of Usb4 devices connected, GetDevices will not throw SDK_NO_DRIVER exception, instead, it will provide the devices.

3.2.3.3 UpdateFirmwareFromContainer

```
public void UpdateFirmwareFromContainer(string filename)
```

Updates device from container in the given filename.

3.2.3.4 ValidateContainer

```
public void ValidateContainer(string containerImagePath)
```

The function to validate the container using USB4 validation on DROM.

3.2.3.5 Fields

Type	Name	Value or Description
String	UUID	The UUID of the device controller
String	ControllerId	The ID of the host controller the device is connected to
UInt32	PortNum	0-based index of the port in the host controller the device is connected to
UInt32	PositionInChain	1-based index of the device position in the port the device is connected to
String	VendorName	Device vendor name
String	ModelName	Device model name
UInt32	VendorId	Device vendor ID (Note: the actual value is bounded to be UInt16)
UInt32	ModelId	Device model ID (Note: the actual value is bounded to be UInt16)

Type	Name	Value or Description
UInt8	ControllerNumber	Controller enumeration out of total number of controllers in the device
UInt8	NumberOfControllers	Total number of controllers in the device
Boolean	Updatable	True if the device is updatable
Byte	LinkSpeed	Device CIO link speed in Gbps
Bool	ConnectedThroughUsb	Indicates if a device is connected through USB.
Bool	IsUsb4Device	Indicates if a device is USB4.
Bool	HasReadDromSupport	Indicates if a device supports DROM read.

3.2.4 TbtException, TbtStatus and Error Handling

The file Exceptions.cs includes the tools this project and the samples use for error reporting.

3.2.4.1 TbtException

All functions in this project use this class for exceptions.

3.2.4.2 TbtStatus

An enum that includes definition for the error codes described above (Return Codes) and additional error codes as follows:

Value	Source	Description
0x200	SDK	General error; used by sample applications for errors that are originated from the driver
0x201	SDK	Internal error; used for notifying about SDK internal coding error
0x202	SDK	No command supplied; used by CLI sample when it runs without any command argument
0x203	SDK	Command not found; used by CLI sample
0x204	SDK	Argument count mismatch; used by CLI sample
0x205	SDK	Invalid host controller ID supplied
0x206	SDK	Invalid device controller UUID supplied
0x207	SDK	No file found in the supplied path for firmware image file
0x208	SDK	Service not found
0x209	SDK	Load host controllers failed

Value	Source	Description
0x20A	SDK	Load devices failed
0x20B	SDK	No host controller found in system
0x20C	SDK	No device found in system
0x20D	SDK	Operation isn't available when the host controller is in safe-mode
0x20E	SDK	Reserved for backward compatibility; not used
0x20F	SDK	This device controller doesn't support device firmware update
0x210	SDK	Reserved for backward compatibility; not used
0x211	SDK	Host/device controller presents an unknown chip
0x212	SDK	The supplied firmware image file is invalid (damaged file)
0x213	SDK	The supplied firmware image file failed validation (incompatible with the host/device controller)
0x214	SDK	The supplied firmware image file is for another hardware generation (incompatible with the host/device controller)
0x215	SDK	The supplied firmware image file is for different port count (incompatible with the host/device controller)
0x216	SDK	The supplied firmware image file can't fit into chip size (incompatible with the host/device controller)
0x217	SDK	Trying to update device controller with a firmware image file intended for host controller
0x218	SDK	Trying to update host controller with a firmware image file intended for device controller
0x219	SDK	Mismatch between the supplied firmware image file and the host/device controller with regarding to PD firmware existence (one has it and one doesn't)
0x21A	SDK	No DROM section found in the supplied firmware image file
0x21B	SDK	Reserved for backward compatibility; not used
0x21C	SDK	The supplied firmware image file is for products of a different vendor than the host/device controller's vendor
0x21D	SDK	The supplied firmware image file is for a different product model than the host/device controller
0x21E	SDK	No matching devices found for the supplied image (used by DeviceFWUTool)
0x21F	SDK	Multiple firmware image files found (used by DeviceFWUTool)
0x220	SDK	The supplied command is not supported on a device
0x221	SDK	Deprectaed method

Value	Source	Description
0x222	SDK	The supplied argument is invalid
0x223	SDK	No DROM section found
0x224	SDK	Native mode mismatch
0x225	SDK	Driver communication error
0x226	SDK	Error during image update
0x227	SDK	DCH driver is missing
0x228	SDK	The image provided is not signed. Therefore rejected.
0x229	SDK	A timeout has expired while FW update was performed. Not thrown on HR FW update.
0x22A	SDK	More than one handle open during FW update.
0x22B	SDK	Driver API unknown.
0x22C	SDK	Operation is not supported.
0x22D	SDK	Invalid container format.
0x22E	SDK	Invalid container input.
0x22F	SDK	Retimer access is not supported by this driver.
0x230	SDK	DROM address is unknown for this device.
0x231	SDK	Retimer not found
0x232	SDK	Image for retimer mismatch
0x233	SDK	Image for device mismatch
0x234	SDK	Number of retimers mismatch

3.2.5 Utilities

This class includes some utility functions for application developer's convenience.

3.2.5.1 GetImageNvmVersion – Deprecated

Note: This method is deprecated and exists only for backward compatibility. Use GetImageFullNvmVersion method instead.

```
public static UInt32 GetImageNvmVersion(string path)
```

Gets the NVM version from the new image (from the binary file in the given path). Returns only the major number of the version.

3.2.5.2 GetImageFullNvmVersion

```
Public static string GetImageFullNvmVersion(string path)
```

Gets the NVM version from the new image (from the binary file in the given path). Returns the version in major.minor format. Used in this sample to display the new image, and allow the user to compare with the existing image version.

3.2.5.3 SafeGetVersion

```
public static string SafeGetVersion(Func<string> func)
```

Allows calling get current version functions in a safe manner which returns "N/A" when in safe mode.

3.2.5.4 GetImageTIPdVersion

```
public static string GetImageTIPdVersion(string path)
```

Reads the PD version from FW image file.

3.2.5.5 GetTIPdInfo

```
public static string GetTIPdInfo(SdkTbtBase controller)
```

This method demonstrate how to use the I2CRead method for a specific TI PD controller It would work with PD controller TPS65982 and TPS65983 (but not limited to).

3.2.5.6 DeviceInformation

```
public class DeviceInformation
{
   public UInt16 VendorId { get; set; }
   public UInt16 ModelId { get; set; }
}
```

Convenience class, to wrap Vendor ID and Model ID for device.

3.2.5.7 GetImageDeviceInformation

```
public static DeviceInformation GetImageDeviceInformation(string path)
```

Reads device information (vendor and model IDs) from FW image file.

3.2.5.8 GetImageOsNativePciEnumerationStatus

```
public static bool GetImageOsNativePciEnumerationStatus(string path)
```

Reads "Native express" status from FW image file.

3.2.5.9 HostUpdateSupported

```
public static bool HostUpdateSupported(string controllerId)
```

Checks if a host controller is supported for FW update.

3.2.5.10 IsSupported

```
public static bool IsSupported(string controllerId)
```

This method is used to make sure this SDK supports the the firmware update through the given host controller.

3.2.5.11 IsIntelDevice

```
public static bool IsIntelDevice(ushort vendorId)
```

Checks if the device is Intel made.

3.2.5.12 IsLegacyDevice

```
public static bool IsLegacyDevice(ushort devid)
```

Checks if the device is TBT3 only.

3.2.5.13 IsLegacyHost

```
public static bool IsLegacyHost(ushort devid)
```

Checks if the host is TBT3 only.

3.2.5.14 RelativeNvmOffsetRetrieveMask

```
public const uint RelativeNvmOffsetRetrieveMask = (1 << 24) - 1;
```

The field is used to read relative nvm offset from image file.

3.2.5.15 RetrieveDeviceIdFromImage

```
public static ushort RetrieveDeviceIdFromImage(string path)
```

Reads device id from FW image file.

3.2.5.16 GetDeviceType

```
public static string GetDeviceType(ushort deviceId)
```

Returns a string representation of device type.

3.2.6 FlashProgress

This class is used to notify an application about firmware update status. It exposes the event with the data described below.

```
public event PropertyChangedEventHandler PropertyChanged
```

This is the event that is raised by this class. User applications should register to this event in order to get progress reports.

```
public UInt32 Progress
```

This is the property to get the information from.

3.2.7 DeviceImageContainer

Class represents a container, which includes GR and DBR NVM images and their details.

3.2.7.1 GetImageFullNvmVersionFromContainer

```
public static byte[] GetImageFullNvmVersionFromContainer(FileStream imageFileStream);
```

Retreives GR NVM version from the container. In case there is no GR in the container, returns [0, 0]

3.2.7.2 GetImageDeviceInformationFromContainer

```
public static List<byte[]> GetImageDeviceInformationFromContainer(FileStream
imageFileStream);
```

Retrieves byte arrays holding vendor and model IDs – [VendorId, ModelId]. In case there is no GR in the container, returns empty List.

3.2.7.3 IsImageContainer

```
public static bool IsImageContainer(string path)
```

Checks if the NVM in the given path is a container.

3.2.7.4 GetDeviceIdFromContainer

```
public static ushort GetDeviceIdFromContainer(FileStream imageFileStream)
```

Reads the device id from the container.

3.2.8 DeviceContainerUtilities

Class to hold useful method to manipulate the container.

3.2.8.1 ExtractNvmDescriptor

```
internal static NvmProperties ExtractNvmDescriptor(FileStream container, uint
offsetNvmProperties)
```

Read nvm descriptor from container (includes pointer to the nvm in container and nvm's size).

3.2.8.2 GetRelativeNvmFromContainer

```
internal static BinaryReader GetRelativeNvmFromContainer(FileStream imageFileStream, int
pointerPosition)
```

This function gets the reader which starts at digital section of the NVM.

3.2.8.3 IsGrPresentInContainer

```
internal static bool IsGrPresentInContainer(FileStream imageFileStream)
```

This function returns true if the container, which represented in imageFileStream, has GR image in it.

3.2.8.4 ExtractDescriptors

```
internal static uint ExtractDescriptors(FileStream container, out NvmProperties GRProperties,
                    out Queue<NvmProperties> RetimersProperties, uint numOfRetimers)
```

This function extracts nvm descriptors for both GR and DBR.

3.2.9 RetimerImageUtilities

Class to hold useful method to manipulate the retimer image.

3.2.9.1 IsDbrImage

```
public static bool IsDbrImage(string path)
```

The function determines if an image with given path is a DBR image.

3.2.9.2 IsDbrImage

```
public static byte ExtractRetimerAddress(string path)
```

This function extracts the retimer SMBUS address from the image with given path. The assumption is that the image is of DBR.

3.2.10 IntelUsb4Validator

Class to use for Intel USB4 devices image validation before FW update.

3.2.10.1 DoesMatch

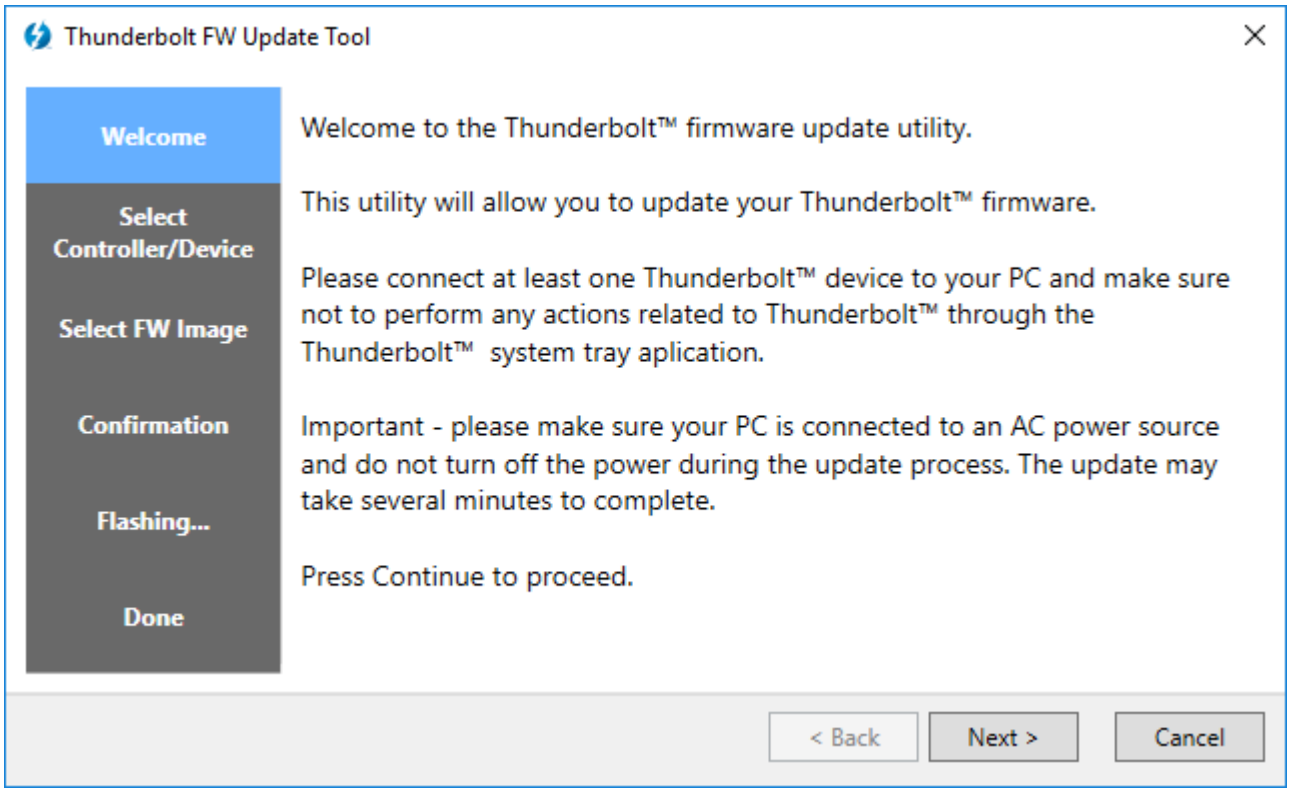
```
public static bool DoesMatch(NvmImageDescriptor desc, IDriverDevice device)
```

Checks if image in the given descriptor matches to the given device.

```
public static bool DoesMatch(NvmImageDescriptor desc, IDriverDevice device, RetimerRoute
route)
```

Checks if image in the given descriptor matches to the retimer with the given route on a given device.

3.3 FwUpdateTool Project



Most of the code in this sample code is UI-related and is not covered by this document.

This sample application is designed for a step by step process to perform the firmware update flow:

1. Select the Thunderbolt host controller or device to be updated
2. Select the new firmware image to be applied
3. Check compatibility between new image and selected Thunderbolt controller configuration
4. Start firmware update process

Note: Device controllers that are part of a multiple-controller device may appear in the list of the controllers followed by their enumeration (e.g. "1/2"). Please take note of the remarks at the end of the document before updating multiple-controller devices.

3.4 FwUpdateCmd Project

```
FW Update CMD - allows user to update the controller firmware.
                Prerequisites: Thunderbolt(TM) software should be fully
                installed and controller is powered.

This sample must run with Administrative privileges

Usage:
FwUpdateCmd EnumControllers
FwUpdateCmd EnumUpdatableDevices
FwUpdateCmd GetTopology
FwUpdateCmd GetCurrentNvmVersion <controller ID / device UUID>
FwUpdateCmd I2CRead <controller ID / device UUID> <port> <offset> <length>
FwUpdateCmd I2CWrite <controller ID / device UUID> <port> <offset> <data>
FwUpdateCmd GetTIPdInfo <controller ID / device UUID>
FwUpdateCmd GetControllerInfo <controller ID>
FwUpdateCmd FWUpdate <controller ID / device UUID> <imagePath>
FwUpdateCmd Help

Parameters:
controller ID - controller ID as returned from EnumControllers or GetTopology commands
device UUID   - device UUID as returned from EnumUpdatableDevices or GetTopology commands
imagePath     - valid NVM image path
port          - port number (1 - based index)
offset        - offset in I2C registers (in hex, with no prefix)
length        - length in bytes to read (decimal)
data          - data to write to an I2C register, in hex format without any delimiters or prefix e.g. AC0F0102. Number of digits
                must be even


Output formats:

EnumControllers - Prints all the controller IDs line by line

EnumUpdatableDevices - Prints each updatable device in a separated line in the following format: UUID VendorID ModelID
                       ControllerNumber/NumberOfControllers

GetTopology - Prints all connected devices in a hierarchical format grouped by controller and port, ordered and numbered by
              position in the chain and includes UUID, vendor name, model name, controller number and number of controllers in
              the device (formatted as X/N), if it's updatable, NVM version and CIO link speed(printed in Gbps). Here is an
              example:
<ControllerID> <NVMversion>
    Port 1:
        <position> <UUID> <VendorName> <ModelName> <ControllerNumber>/<NumberOfControllers> <Updatable> <NVMversion> <LinkSpeed>
    Port 2:
        <position> <UUID> <VendorName> <ModelName> <ControllerNumber>/<NumberOfControllers> <Updatable> <NVMversion> <LinkSpeed>
<ControllerID> <NVMversion>
    Port 2:
        <position> <UUID> <VendorName> <ModelName> <ControllerNumber>/<NumberOfControllers> <Updatable> <NVMversion> <LinkSpeed>

I2CRead - Prints the content of an I2C register

I2CWrite - Writes data into I2C register

The following is TI PD controller specific
GetTIPdInfo - Prints TI PD version, CUSTUSE and CustomerVersion registers
```

This sample application is command line based and was designed with automation in mind, so output would be easily parsable and reused as parameter of the different implemented commands.

It provides the following commands:

1. EnumControllers – Enumerate Thunderbolt host controllers
2. EnumUpdatableDevices – Enumerate Thunderbolt updatable devices.
3. GetTopology – Show all devices in a tree-like format; only host controllers and ports that have devices connected to them will be shown
4. GetCurrentNvmVersion – Print current NMV version (in major.minor format) for a given Thunderbolt host/device controller
5. I2CRead – Prints I2C register content for a given Thunderbolt host/device controller.
6. I2CWrite – Writes to I2C register content on a given Thunderbolt host/device controller.
7. GetTIPdinfo – Prints TI specific wanted I2C registers.
8. GetControllerInfo – Print information on a given Thunderbolt host controller
9. FWUpdate – Perform firmware update on a given Thunderbolt host/device controller
10. Help – Print help about the available commands and output formats of the various host and device controller listing commands

Since this sample application is based on the same Thunderbolt API wrapper as the GUI based sample, it also performs the same kind of compatibility check between the new image and the selected Thunderbolt controller configuration before applying the new NVM image.

This sample uses application return value (ERRORLEVEL) to notify any error condition as described above. It also prints the error code number, the enum entry name and detailed description if found.

3.4.1 Remarks

- This sample must run with Administrative privileges in order to be able to use the GetCurrentFullNvmVersion and FW update commands.
- The controller ID format includes command-line special characters (e.g. '&'), so it must be quoted to pass it as an argument.
- The user is advised to update multiple-controller devices starting with the device controller that is farthest from the host, ending with the device controller that is closest to the host.

3.5 DeviceFWUTool

This is a sample application, which allows a user to update the compatible device image.

4 Appendix

The SDK contains the following libraries:

File name / version	version
FwUpdateDriverApi.dll	1.13.0.0
FwUpdateApi.dll	1.13.0.0