



# **TL-USBDFU Solution**

## **User Manual**

Version: 2.20.0

Date: 4 November 2019

ToriLogic GmbH & Co. KG  
Werner-von-Siemens-Str. 2  
98693 Ilmenau  
Germany  
+49 3677 8462 0  
<http://www.torilogic.com>



## Contents

1	TL-USBDFU Overview .....	6
1.1	Supported Windows Operating Systems .....	6
1.2	Supported macOS Systems.....	6
1.3	Supported Linux Systems.....	6
1.4	TL-USBDFU Features .....	6
1.5	Windows-Specific Features.....	6
1.6	Software Architecture.....	7
1.7	TL-USBDFU License Mechanism .....	7
1.7.1	License Data Record.....	7
1.8	Evaluation Version .....	8
1.9	Deliverables.....	8
1.9.1	Driver Install Package .....	8
1.9.2	Driver Support Kit (DSK) .....	8
2	USB Device Implementation.....	9
2.1	Device Requirements Summary .....	9
2.2	Unique USB VID and PID .....	9
2.2.1	Vendor ID (VID) .....	9
2.2.2	Product ID (PID) .....	10
2.2.3	ToriLogic-provided VID and PID .....	10
2.3	Distinct PIDs in Application and DFU Mode .....	10
2.4	Consistent USB Serial Number in Application and DFU Mode .....	10
2.5	DFU Device Class v1.1 Compliance.....	11
2.6	Driverless Operation on Windows 10 .....	11
3	TL-USBDFU Configuration and Customization.....	12
3.1	Firmware Update Publishing .....	12
3.1.1	Ship Firmware Image Files and DFU Utility Separately (not recommended) .....	12
3.1.2	Ship Firmware Image Files and DFU Utility as One Package (recommended).....	12
3.2	Windows Device Driver.....	13
3.3	TL-USBDFU DLL Configuration on Windows.....	13
3.4	GUI Utility Configuration (Windows and macOS).....	14
3.4.1	Windows GUI Utility Configuration.....	16
3.4.2	macOS GUI Utility Configuration .....	17
3.5	GUI Utility Localization (Windows and macOS) .....	19

3.5.1	Windows GUI Utility Localization .....	20
3.5.2	macOS GUI Utility Localization.....	20
3.5.3	Language-specific Text Files Syntax .....	22
4	Windows Driver Installer .....	23
4.1	Driver Setup Files.....	23
4.2	Self-extracting Driver Install Package.....	23
4.3	Driver Package Delivery to End Users.....	23
4.3.1	Why an exe should be Delivered Instead of a zip?.....	24
4.4	Integration with an Overall Software Installer .....	24
4.5	Driver Installer Silent Mode (Non-Interactive).....	24
5	TL-USBDFU Software Development Kit (SDK) .....	25
6	Analyzing Problems on Windows .....	26
6.1	Driver Installer Problem.....	26
6.1.1	Installer Log File .....	26
6.1.2	Setupapi Log Files.....	26
6.2	Error Code in Device Manager.....	27
6.2.1	Device Enumeration Failed .....	27
6.2.2	Code Signing Issue .....	27
7	Analyzing Problems on macOS.....	28
8	Known Issues .....	29
8.1	Windows 7 requires hotfix to accept SHA-256 code signature.....	29
8.2	Untrusted Publisher pop-up dialog appears on Windows 7.....	29

## References

- [1] Universal Serial Bus Specification, Revision 2.0, April 27, 2000, usb\_20.pdf,  
[http://www.usb.org/developers/docs/usb20\\_docs/usb\\_20\\_042814.zip](http://www.usb.org/developers/docs/usb20_docs/usb_20_042814.zip)
- [2] USB ENGINEERING CHANGE NOTICE, Interface Association Descriptors,  
InterfaceAssociationDescriptor\_ecn.pdf,  
[http://www.usb.org/developers/docs/usb20\\_docs/usb\\_20\\_042814.zip](http://www.usb.org/developers/docs/usb20_docs/usb_20_042814.zip)
- [3] Universal Serial Bus 3.1 Specification, Revision 1.0, July 26, 2013, USB\_3\_1\_r1.0.pdf,  
[http://www.usb.org/developers/docs/usb\\_31\\_031114.zip](http://www.usb.org/developers/docs/usb_31_031114.zip)
- [4] Universal Serial Bus Device Class Specification for Device Firmware Upgrade,  
Version 1.1, August 5, 2004, DFU\_1.1.pdf,  
<https://www.usb.org/document-library/device-firmware-upgrade-11-new-version-31-aug-2004>

## **1 TL-USBDFU Overview**

ToriLogic's TL-USBDFU is a host-side, customizable, cross-platform software solution for device firmware upgrade via USB. The solution works with devices that are compliant to the USB Device Firmware Upgrade (DFU) v1.1 specification [4].

### **1.1 Supported Windows Operating Systems**

- Windows 10 (32-bit and 64-bit)
- Windows 8.1 (32-bit and 64-bit)
- Windows 8 (32-bit and 64-bit)
- Windows 7 (32-bit and 64-bit)
- Windows Server 2016 (64-bit)
- Windows Server 2012 R2 (64-bit)
- Windows Server 2012 (64-bit)
- Windows Home Server 2011 (64-bit)
- Windows Server 2008 R2 (64-bit)

### **1.2 Supported macOS Systems**

- macOS 10.15
- macOS 10.14
- macOS 10.13
- macOS 10.12

### **1.3 Supported Linux Systems**

- Linux kernel 4.x and later (32-bit and 64-bit)

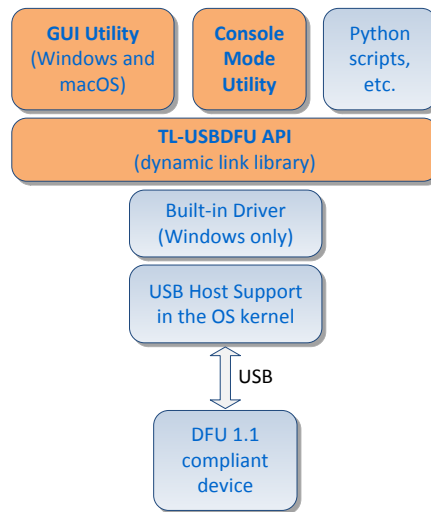
### **1.4 TL-USBDFU Features**

- Uniform C function style API implemented in a dynamic link library
- C++ wrapper classes for easy integration into applications implemented in C++
- API design allows easy integration into .net and C#, and scripting languages such as Python
- Software development kit including sample source code available
- Full customization and branding supported
- Full source code available on request (subject to a separate license agreement)

### **1.5 Windows-Specific Features**

- Driverless operation supported on Windows 10
- Microsoft compliant Windows USB device driver (WinUsb based)
- Conforms with current WHQL and Hardware Lab Kit (HLK) requirements
- Optional driver installer for reliable installation, update and uninstallation

## 1.6 Software Architecture



**Figure 1 TL-USBDFU Software Architecture**

## 1.7 TL-USBDFU License Mechanism

The TL-USBDFU dynamic link library implements a license mechanism that is designed to protect ToriLogic’s intellectual property and also to protect the customer’s investment. The mechanism ensures that a licensed copy of the driver can be used with the customer’s devices only, and cannot be pirated and used with arbitrary devices.

For a customer or end user who received a licensed copy of the driver, the technical implementation of the license mechanism is imperceptible. The license data is encoded in a “.ini” file. It is not required to enter any license keys during installation time or runtime. Also there is no online license check. The driver and the installer do not transfer any data online.

### 1.7.1 License Data Record

For each copy of the TL-USBDFU library, ToriLogic creates a license data record which reflects the requirements the customer specified when the license was purchased. The license data record is contained in a “.license.ini” file. On Windows the file resides next to the TL-USBDFU dynamic link library, on macOS it resides in the subfolder “Contents/Resources” of the application package. The record is protected against tampering by means of cryptographic checksums. The dynamic link library loads and starts only if a valid license record is present.

Basically, the license data record encodes the following information:

- A set of USB vendor IDs (VID) and product IDs (PID). A given device must match one of the licensed VID/PID combinations.
- Release Version. A given license data record cannot be reused with a different library version. Together with a new library release, ToriLogic provides a fresh license data record.

## 1.8 Evaluation Version

ToriLogic offers a free evaluation version of the TL-USBDFU solution. This version comes with a temporary license which expires after 60 days. During the evaluation period the solution works without any restrictions. After the evaluation license period has expired, the software stops working.

The evaluation copy is restricted to the VID/PID combinations provided to ToriLogic when the evaluation version was requested. If you need to add another PID during the evaluation phase, please request a new evaluation copy.

## 1.9 Deliverables

To a licensee of TL-USBDFU, ToriLogic delivers various software packages depending on the components the license includes. The content of each package is described subsequently.

### 1.9.1 Driver Install Package

Example: ToriLogic\_USBDFU\_v1.11.0\_2018-11-21\_setup.exe

This is the driver install package to be shipped to end users. It is a self-extracting zip archive that automatically launches the driver installer after extraction. The driver and the self-extracting EXE is signed with ToriLogic's EV code signing certificate to ensure that the user can download the file from a web site without problems.

Note that the driver installer is an optional component. If the installer is not included then this package is simply a self-extracting zip archive which does not use the file name suffix `_setup`.

### 1.9.2 Driver Support Kit (DSK)

Example: ToriLogic\_USBDFU\_v1.11.0\_DSK\_2018-11-21.exe

The DSK contains all components required by a licensee to integrate the TL-USBDFU driver with a custom application. It is a self-extracting zip archive which contains:

- Documentation (user manual, revision history, etc.)
- API DLL executables required for integration
- sample application executables
- TL-USBDFU driver and API debug builds for diagnostics
- Driver setup files tree for release and debug build

The DSK must **not** be shipped to end users "as-is". However, the licensee is allowed to ship some of the included components, if required.



## 2 USB Device Implementation

This section provides information on the functionality to be implemented in a USB device to work properly with ToriLogic's TL-USBDFU solution.

### 2.1 Device Requirements Summary

Below is a summary of requirements to be implemented in the device firmware. A more detailed discussion can be found in subsequent sections.

- 1) For each new product to be released to the market, a unique vendor ID (VID) and product ID (PID) tuple must be used. This is important because on Windows driver installation is based on VID/PID. See also 2.2.
- 2) A distinct PID must be used for Application and DFU mode. See also 2.3.
- 3) A product must either implement no USB serial number or a unique per-instance USB serial number. But in either case the reported USB serial number must be consistent in Application and DFU Mode. See also 2.4.
- 4) The firmware implementation must be compliant with the DFU specification v1.1 [4]. See also 2.5.
- 5) For driverless operation on Windows 10, the device must implement a Microsoft OS descriptor. See also 2.6.

### 2.2 Unique USB VID and PID

Every shipping USB product is uniquely identified by a tuple of two 16-bit integer numbers: vendor ID (VID) and product ID (PID). Each number ranges from zero to 65536 (0x0000...0xFFFF). A USB device reports these numbers through its device descriptor in the fields `idVendor` and `idProduct` (see [1] for details).

Operating systems (Windows, macOS, Linux) use the VID/PID tuple to locate the matching device driver, and to store device-specific configuration information (e.g. last volume set). Software applications such as configuration tools, firmware upgrade utilities, etc. typically rely on the VID/PID tuple to identify the device they have been designed for.

If two different products share the same VID/PID tuple then an operating system will potentially load the wrong device driver, and another product's applications and utilities will possibly try to communicate with the device, perform a firmware update etc. This leads to bad user experience and, in a worst-case scenario, can damage the device.

Hence it is mandatory to **ensure that a unique VID/PID tuple is assigned** to every distinct USB device type (model) that is shipping. Details on how VID and PID assignment is done in practice are given in the following sections.

See also section 2.3 below.

#### 2.2.1 Vendor ID (VID)

The Vendor ID is globally unique and typically identifies the manufacturer or distributor of the device. To ensure uniqueness globally, VIDs are managed and assigned by the USB Implementers Forum organization (<http://www.usb.org>). To receive a VID, you have to contact the USB Implementers Forum, see <http://www.usb.org/developers/vendor/>.

Any shipping product must **use the official VID** assigned by the USB Implementers Forum to the product's manufacturer.

In some cases it is possible to use the VID of a silicon vendor. However, this requires registration with the silicon manufacturer or distributor to receive a unique PID. Never re-use the VID/PID of an evaluation board or reference design because other product developers will potentially do this as well!

### 2.2.2 Product ID (PID)

The owner of a VID manages the corresponding PID range. PIDs can be assigned arbitrarily but need to be chosen in such a way that every shipping USB product uses a unique VID/PID tuple.

Make sure you **choose two unique PIDs for each new device model, one for APP mode, one for DFU mode** and carefully keep records on PID assignments. See also section 2.3 below.

### 2.2.3 ToriLogic-provided VID and PID

ToriLogic owns an officially registered VID. On request ToriLogic provides licensees of the TL-USBDFU solution with a unique PID which can be used in combination with that VID. For licensees, this service is free of charge but requires registration with ToriLogic.

## 2.3 Distinct PIDs in Application and DFU Mode

The DFU specification [4] defines two different modes of operation:

- APP mode (or application mode): The device runs the normal firmware image.
- DFU mode (or bootloader mode): The device runs a bootloader image or similar.

For the TL-USBDFU solution to work correctly on Windows, it is required that a distinct PID is used in each mode. Hence it is required to reserve two different PID values for a given product, one is to be reported by the firmware in APP mode, while the other one is to be reported by a bootloader when in DFU mode.

This requirement originates from the behavior of Microsoft's built-in USB composite driver. In APP mode this driver is required to split the device into two (or more) separate logical devices, one for the normal device functionality, and one for DFU. But in DFU mode the composite driver does not load (because the device exposes one USB interface only). Instead the TL-USBDFU driver must be loaded. Switching between composite driver and DFU driver works only if Windows treats each mode as a distinct device which can be achieved by distinct PIDs.

Note that this requirement is also mentioned (as a recommendation) in the DFU specification [4], section "2. Overview".

## 2.4 Consistent USB Serial Number in Application and DFU Mode

When the device switches between APP and DFU mode, an application must be able to identify the device instance after the device detached and re-attached itself. To achieve this, a USB serial number must be implemented consistently in both APP and DFU mode. The guidelines are as follows:

- If no USB serial number is reported in the device descriptor in APP mode (`iSerialNumber` field is set to zero) then no serial number should also be reported in DFU mode. Windows emulates a serial number based on the hub port number in this case.
- If a USB serial number is reported in the device descriptor in APP mode then exactly the same serial number should also be reported when the device operates in DFU mode.

## 2.5 DFU Device Class v1.1 Compliance

Generally, to work with the TL-USBDFU solution a device must be compliant with the DFU device class specification v1.1 [4]. Specifically, the following details need to be considered:

- The field `bcdDFUVersion` in the DFU Functional Descriptor must be set to 0x0110.
- Bit 3 (`bitWillDetach`) must be set in the `bmAttributes` field in the DFU Functional Descriptor. This indicates that the device detaches and switches to DFU mode when it receives a `DFU_DETACH` request.
- Bit 2 (`bitManifestationTolerant`) should be set in the `bmAttributes` field in the DFU Functional Descriptor. This indicates that the device is able to communicate via USB after the image download is finished.
- Bit 0 (`bitCanDnload`) must be set in the `bmAttributes` field in the DFU Functional Descriptor to indicate that firmware image download is supported.
- The field `bInterfaceProtocol` in the DFU Interface Descriptor must be set correctly to indicate the run mode of the device. `bInterfaceProtocol=1` indicates Application mode, `bInterfaceProtocol=2` indicates DFU mode.

The TL-USBDFU library does never issue a USB Reset. Hence the `bitWillDetach` feature must be implemented.

To switch from APP mode to DFU mode, the TL-USBDFU library issues a `DFU_DETACH` request.

To switch from DFU mode back to APP mode after firmware image upload is finished, the library also issues a `DFU_DETACH` request. This is an extended use of the `DFU_DETACH` request and is actually an extension to the DFU specification which states that USB Reset is to be used for mode switching. However, USB Reset is problematic to implement on host side and can cause side effects. Hence the TL-USBDFU library does not use USB Reset.

If multiple image targets (e.g. FLASH areas) are implemented in the device then these must be addressable via alternate settings in the DFU Mode Interface descriptor. See [4] section 4.2.3.

## 2.6 Driverless Operation on Windows 10

On Windows the TL-USBDFU solution is based on Microsoft's in-box WinUSB device driver. On Windows 10 (Windows 8 and later, to be exact) this driver is available as a built-in class driver. For details, check out

<https://docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/automatic-installation-of-winusb>

The WinUSB driver can be loaded automatically without any user intervention, if the device implements a Microsoft OS descriptor and reports a compatible ID "WINUSB" through this descriptor. For details, refer to the following documentation:

<https://docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/automatic-installation-of-winusb#how-to-configure-a-winusb-device>

<https://docs.microsoft.com/en-us/windows-hardware/drivers/usbcon/microsoft-defined-usb-descriptors>

Note that it is **mandatory** to generate a unique device interface GUID for the given device. Refer to the "DeviceInterfaceGUID" property description in the above documentation. The device interface GUID is used by the TL-USBDFU library to locate and open the device instance. For details on dynamic link library configuration, see 3.3.

## 3 TL-USBDFU Configuration and Customization

### 3.1 Firmware Update Publishing

Various approaches for deploying firmware updates in the field will be discussed in the next subsections.

#### 3.1.1 Ship Firmware Image Files and DFU Utility Separately (not recommended)

With this approach, the firmware for a specific product “MyAudioDevice” is provided (e.g. through web download) as a plain binary file, e.g. MyAudioDeviceFirmware\_v1.10.bin. The DFU utility is distributed separately either standalone or as part of another software package. To perform a firmware upgrade, the user has to download the firmware .bin file from the web, launch the DFU utility, pick the .bin file from a local directory and start the download.

This approach is **not recommended** by ToriLogic for the following reasons:

- If you offer multiple products, or different hardware revisions of one product, there is a chance that the user downloads the wrong firmware file from the web.
- When the user browses for the firmware file on the local machine, there is a chance that she/he picks an incorrect file.

The DFU standard does not define a mean to check whether a given file is a valid firmware image. So if the user made a mistake, she/he could send an arbitrary file to the device. If the device has no mechanism to detect an invalid firmware image, this could render the device useless, or damage it.

#### 3.1.2 Ship Firmware Image Files and DFU Utility as One Package (recommended)

For rolling out a firmware update, ToriLogic recommends to ship a preconfigured software package which consists of the following components:

- Customized TL-USBDFU dynamic link library, see section 3.3
- Customized DFU application, including .xml configuration file and optional GUI language files, see sections 3.4 and 3.5
- One or more firmware image files (.bin or .timg) which are referenced in the XML configuration file

The XML file is preconfigured to list the device types supported by this package and the corresponding firmware files, see 3.4.1 for details. For a given device, the DFU utility will automatically pick a firmware file from the package, or refuse updating the device if the model is not listed in the XML file. The user does not have to browse for a file. So there is no risk of making mistakes.

Note that one firmware update package can contain updates for several distinct products (e.g. a product family) or different revisions of one product.

### 3.2 Windows Device Driver

On Windows, direct access to USB devices from user mode is not possible. A kernel-mode device driver is required. A built-in driver can be used for that purposes. But depending on Windows version and capabilities implemented in the device firmware, this driver either loads automatically or must be installed manually.

The following table summarizes the requirements.

	Windows 7 / 8 / 8.1	Windows 10
Device does not implement MS OS descriptor	Device driver must be installed manually, see section 4	
Device implements MS OS descriptor and “WINUSB” compatible ID	Device driver must be installed manually, see section 4	Driverless operation, see section 2.6

### 3.3 TL-USBDFU DLL Configuration on Windows

The TL-USBDFU dynamic link library (DLL) must be able to locate, identify and open the device instances to work with. On Windows this requires a device interface GUID which serves as a unique identifier that is exposed by the device instances available in the system.

The device interface GUID is specified in a “.config.ini” file that resides next to the TL-USBDFU DLL. The DLL is able to enumerate device instances only if a valid “.config.ini” file is present.

The syntax of the “.config.ini” file is as follows:

```
[DriverInterface]
InterfaceGUID = {D3317FB4-900F-4018-AEAE-0522454D1EBA}
```

The shown GUID is an example and will be replaced by a fresh GUID for each customized copy of the TL-USBDFU software package.

### 3.4 GUI Utility Configuration (Windows and macOS)

The DFU utility can be configured through an UTF-8 encoded XML file. The format and the features supported by this XML configuration file are almost identical on Windows and macOS. There are only very few platform-specific configuration options which are ignored by the other platform. Therefore, the XML configuration file only needs to be created once and can then be used for both the DFU utility on Windows and macOS. The following example file shows all available configuration settings. Some are optional.

```
<?xml version="1.0" encoding="utf-8" ?>
<DFU FormatVersionMajor="1" FormatVersionMinor="0">
  <DfuDllFileName>MyFirmwareUpdate.dll</DfuDllFileName>
  <SupportedDevices>
    <Device UseXmosDfuExtensions="True">
      <AppMode VID="0x152a" PID="0x020a"/>
      <DfuMode VID="0x152a" PID="0x020b"/>
      <Firmware ImageType="TLBinary" Target="0">MyFirmwareDevX_v1_12.tlimg</Firmware>
    </Device>
    <Device>
      <AppMode VID="0x152a" PID="0x021a"/>
      <DfuMode VID="0x152a" PID="0x021b"/>
      <Firmware MajorVersion="2" MinorVersion="18">MyFirmwareDevY_v2_18.bin</Firmware>
    </Device>
  </SupportedDevices>
  <CheckFirmwareVersion>Warn</CheckFirmwareVersion>
  <AllowToBrowseForFirmwareFile>True</AllowToBrowseForFirmwareFile>
  <AllowToAbortUpgrade>False</AllowToAbortUpgrade>
  <DisplayLanguageSelectionDialog>True</DisplayLanguageSelectionDialog>
</DFU>
```

Note that the XML file uses UTF-8 encoding. If you are using Windows Notepad for editing, set Encoding to UTF-8 in the "File - Save As" dialog.

The following describes all the XML elements and attributes that are relevant to the configuration. Unless explicitly stated as optional, all elements described are required. All other elements not mentioned must be adopted unchanged.

#### FormatVersionMajor/ FormatVersionMinor

Major and minor version of the XML file format. These attributes are not edited when the configuration file is customized. However, you need to know their meaning to know if old configuration files can continue to be used when updating the DFU utility. Old configuration files are compatible as long as the format version of the configuration file belonging to the current DFU Utility has not changed or only the minor version has been increased. Using the new XML format is optional as long as the new included features are not needed. However, the new format is mandatory if incompatible changes have been made. This is the case when the major version has been increased.

#### DfuDllFileName

Name of the TL-USBDFU dynamic link library. The name is required on Windows. It is ignored on macOS.

#### SupportedDevices

List of all supported device models. The list can contain any number of different models. At least one model is required.

#### Device

Supported device model.

**UseXmosDfuExtensions**

This attribute is optional. The default value is False. It must be set to true if the DFU functionality of the firmware installed on the device uses XMOS-specific DFU protocol extensions.

**AppMode**

Collection of attributes by which the device model is recognized when running in application mode.

**DfuMode**

Collection of attributes by which the device model is recognized when running in DFU (bootloader) mode.

**VID**

USB vendor ID of the device model. The format of the ID is hexadecimal with 0x prefix.

**PID**

USB product ID of the device model. The format of the ID is hexadecimal with 0x prefix.

**Firmware**

Firmware used to upgrade the device model. It is optional. It can be omitted if the configuration allows the user to browse for firmware files. If omitted, even though the user cannot search for firmware files, only a message appears stating that there is no firmware available for the device. This is useful, for example, if you want to tell the user that the device has been detected, but no upgrade is supported for this model.

**ImageType**

Format type of firmware image. The attribute is optional. The following values are allowed:

- RawBinary: Plain binary firmware image file.
- TLBinary: Firmware image with a format specified by ToriLogic.

If the format type is not defined, it will be derived from the file extension of the firmware file. If the extension is ".tling", the ToriLogic format is assumed, otherwise the binary format.

**MajorVersion/MinorVersion**

Major and minor version of the firmware.

For the common image format type (RawBinary), these version attributes are mandatory, if a firmware version check should be performed (see CheckFirmwareVersion).

For the ToriLogic image format type (TLBinary), these attributes are rejected, if a firmware version check should be performed (see CheckFirmwareVersion), because the version information is included in the image.

**Target**

Specifies the target of the firmware upgrade. The meaning of the value is device-specific. For example, a value of 0 could represent the main executable image in internal FLASH and a value of 1 could identify an external EEPROM. The value is optional. The default value is 0.

**CheckFirmwareVersion**

Defines whether the application should compare the version of the firmware provided for a device with the version of the firmware installed on the device. It also determines how the application should behave according to the result of the comparison. Allowed values are:

- No: No check will be performed. This is the default value.
- Warn: The check will be performed. If the firmware installed on the device is newer, the user will receive a warning and must explicitly agree to replace it with the older firmware or cancel the download.
- Block: The check will be performed. If the firmware installed on the device is newer, the user will be notified and informed that no upgrade is required. The firmware download is disabled in this case.

Note: If the application allows the user to browse for a firmware file (see below) and the selected file does not have the ToriLogic image format, the check will not be performed even if it is enabled because the firmware version is not available.

**AllowToBrowseForFirmwareFile**

True, if the application should allow the user to browse for a firmware file, false (default), if not. Browsing is only recommended for developers or in production. For end users, the firmware should be defined in the configuration for each device model and browsing should be disabled.

**AllowToAbortUpgrade**

True, if the user should be allowed to cancel a running firmware download, false (default), if not. An abort should only be allowed if the device can handle this case correctly and there are reasons to cancel the download. This will probably only be useful in rare cases during development. Generally, however, there is no need. The application uses generous timeouts to ensure that the upgrade finishes after a finite time.

**DisplayLanguageSelectionDialog**

True if you want to display a dialog before startup that allows the user to select a language supported by the application, false (default), if not. Normally, this dialog does not need to be displayed. The application automatically selects from the supported languages of the application the one that best suits the user's system preferences (see also section 3.5).

**3.4.1 Windows GUI Utility Configuration**

The configuration requirements for the Windows GUI application are summarized below.

The TL-USBDFU dynamic link library (.dll) is expected in the same directory as the application's .exe file. The value of the `DfuDllFileName` element in the XML configuration file must be set to the TL-USBDFU DLL filename.

A license file must be present and must match the .dll name plus the ".license.ini" extension. For example, if the library is named "MyFirmwareUpdate.dll", the license file must be named "MyFirmwareUpdate.dll.license.ini".

A configuration file must be present and must match the .dll name plus the "config.ini" extension. For example, if the library is named "MyFirmwareUpdate.dll", the license file must be named "MyFirmwareUpdate.dll.config.ini".



The XML configuration file must reside in the same directory as the application and must use the same file base name. For example, if the application's exe name is "MyFirmwareUpdate.exe" then the XML file in the same directory is "MyFirmwareUpdate.xml".

Any firmware files referenced in the XML configuration file must reside in a subfolder of the application's directory. The subfolder name is the application base name plus ".firmware" extension. For example, if the application name is "MyFirmwareUpdate.exe" then the subfolder name is "MyFirmwareUpdate.firmware" and a firmware file path would be "MyFirmwareUpdate.firmware\firmware.bin", for example.

### 3.4.2 macOS GUI Utility Configuration

The application is signed and notarized. If it is started, an indication of the operating system for safety appears. If you continue and run the application, this note will not appear again on this machine. If changes are made to the application configuration, this invalidates the signature and notarization. However, because signature was checked initially, the modified application still works on the given machine. But due to the invalid signature the modified application cannot be started on another computer. So before it can be distributed, it must be signed and notarized again.

The configuration requirements for the macOS application are summarized below.

The XML configuration must be located in the application package in the subdirectory "Contents/Resources". The name must be "Config.xml".

The file "Info.plist" is located in the application package in the subdirectory "Contents". The file contains strings that are used by the system in the desktop menu of the application, the application name and information in the Finder, the About dialog and so on. Some strings are already displayed before the application is started.

The file "Info.plist" contains entries of the form `<key>...identifier...</key>`. These are identifiers for application settings. Below each key entry of interest is an entry of the form `<string>...value...</string>`. It contains the value of the setting. The values of the following settings are relevant:

- CFBundleDisplayName: Short application name.
- CFBundleName: Short application name.
- CFBundleExecutable: The file name of the application, without extension.
- CFBundleIdentifier: Application identifier that uniquely identifies the app throughout the system. The string must be a uniform type identifier (UTI) that contains only alphanumeric characters (A-Z, a-z, 0-9), hyphen (-), and period (.). The string should be in reverse-DNS format.
- NSHumanReadableCopyright (optional): A human-readable copyright notice for the application.

Note: The values of the keys NSHumanReadableCopyright, CFBundleDisplayName and CFBundleName can be localized (see explanations to the file InfoPlist.strings in section 3.5.2).

The application package contains a license file for the TL-USBDFU dynamic link library. The name of the license file is "libtlusbdfuapi.dylib.license.ini". It is located in the subfolder "Contents/Resources".

Any firmware files referenced in the `Config.xml` configuration file must reside in the "Contents/Resources/Firmware" subfolder of the application package.

The application comes with a sample icon. This icon can either be removed or replaced by a custom one. To remove it, just delete the files `AppIcon.icns` and `Assets.car` in the subdirectory "Contents/Resources" of the application package. The application then uses the standard icon for Mac applications.

To replace the icon, the following steps are necessary:

- First you have to create your own icon. The icon should have the maximum required resolution of 1024 x 1024 pixels.
- This icon must now be provided in additional sizes and resolutions, see App Icon here: <https://developer.apple.com/design/human-interface-guidelines/macos/icons-and-images/app-icon>
- In addition, .json files are needed that describe the icon variants. For the sake of simplicity, you can use a suitable tool, for example the Asset Catalog Creator from the Mac App Store. As a result, the following directory and files should be created:

```
Assets.xcassets
|--- Contents.json
|--- [AppIcon.appiconset]
|    |--- AppIcon-UsbDfu16x16.png
|    |--- AppIcon-UsbDfu16x16@2x.png
|    |--- AppIcon-UsbDfu32x32.png
|    |--- AppIcon-UsbDfu32x32@2x.png
|    |--- AppIcon-UsbDfu128x128.png
|    |--- AppIcon-UsbDfu128x128@2x.png
|    |--- AppIcon-UsbDfu256x256.png
|    |--- AppIcon-UsbDfu256x256@2x.png
|    |--- AppIcon-UsbDfu512x512.png
|    |--- AppIcon-UsbDfu512x512@2x.png
|    |--- Contents.json
```

Possibly, the files created by the tool must be renamed. Note that you also have to edit the file names in the .json file in this case.

- The generated files must now be compiled. This requires a command-line tool that comes with Xcode. Xcode can also be downloaded from the Mac App Store. After installing Xcode, open a terminal and execute the following command:

```
xcrun actool "[ASSET_PATH]/Assets.xcassets" --minimum-deployment-target
10.12 --platform macosx --app-icon AppIcon --output-partial-info-plist
"[OUT_PATH]/Info.plist" --compile "[OUT_PATH]"
```

[ASSET\_PATH] must be replaced with the path of the folder Assets.xcassets.

[OUT\_PATH] must be replaced by the path of the desired output folder.

The result should be the following files: `AppIcon.icns`, `Assets.car` and `Info.plist`.

- Copy the files `AppIcon.icns` and `Assets.car` to the subdirectory "Contents/Resources" of the application package. The file `Info.plist` will not be copied!

Note: The Finder in macOS may not display all changes, as it caches information such as the application name, icon, and so on. To clear this cache and to refresh the display, the following command can be executed in the terminal:

```
touch [path and name of the app]
```

### 3.5 GUI Utility Localization (Windows and macOS)

The DFU utility supports a mechanism to load language-specific text strings for the user interface. The mechanism is based on simple UTF-8 encoded text files provided with the application. Each supported language or language variant is represented by its own file. Depending on the platform, these files are named and deployed differently. However, their content is the same for all platforms. The translation of English texts into the desired language is therefore only required once.

A language or language variant is identified by a language designator, and optional by additional script and region designators. The following sources provide more information about this topic:

- Language and Locale IDs  
(<https://developer.apple.com/library/archive/documentation/MacOSX/Conceptual/BPInternational/LanguageandLocaleIDs/LanguageandLocaleIDs.html>)
- Codes for the Representation of Names of Languages  
([https://www.loc.gov/standards/iso639-2/php/English\\_list.php](https://www.loc.gov/standards/iso639-2/php/English_list.php))
- ISO 3166-1  
([https://en.wikipedia.org/wiki/ISO\\_3166-1](https://en.wikipedia.org/wiki/ISO_3166-1))
- Windows Language Code Identifier (LCID) Reference  
([https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-lcid/70feba9f-294e-491e-b6eb-56532684c37f](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-lcid/70feba9f-294e-491e-b6eb-56532684c37f))
- Windows Language Code Identifier (LCID) Reference -> Product Behavior  
([https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-lcid/a9eac961-e77d-41a6-90a5-ce1a8b0cdb9c](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-lcid/a9eac961-e77d-41a6-90a5-ce1a8b0cdb9c))

Examples of identifiers for languages or language variants:

en-US	for English (United States)
en-GB	for English (United Kingdom)
de	for German
zh-Hans	for Chinese (Simplified)
zh-Hans_HK	for Chinese (Simplified, Hong Kong)

When the application starts, it first checks the system language(s) used by the current user. It then selects the language supported by the application according to the following algorithm:

- If there is a supported language whose language, script and region identifiers exactly match the system language, it will be chosen.
- Otherwise, all supported languages and language variants are searched whose language identifier matches that of the system language. It then selects the language whose optional script and region identifiers match best, if any.
- If no suitable language could be selected, all supported language variants are searched for with English language identifier ( 'en' ). The best matching variant is chosen, taking into account region and script identifiers of the system language, if any.
- If then still no language could be selected, because the application does not provide English texts, the application uses the built-in English text strings.

Note: Normally this algorithm should be sufficient to select a suitable language. However, the application can also be configured to display a language selection dialog that enables the user to choose a GUI language.

### 3.5.1 Windows GUI Utility Localization

All language files are located in a subdirectory of the folder where the application is located. The subdirectory uses the base name of the application plus ".strings" extension. For example, if the name of the application is "MyFirmwareUpdate.exe" then the subdirectory of the corresponding language files "MyFirmwareUpdate.strings".

Note that the ".strings" subdirectory is optional. If it is not present then the application uses built-in English strings as mentioned above.

The name of a language file is composed of the identifier for the language and the extension ".txt" (e.g. en-US.txt). According to the example above, the sub-path of the language file would be "MyFirmwareUpdate.strings\en-US.txt"

To add a new language, follow these steps:

- Create your target text file as a copy of the original en.txt. For example, copy en.txt to zh-Hans.txt.
- Send the new file (which still contains the English strings) to a translator. When modifying the text file, the translator should follow the rules described in section 3.5.3.
- Place the file with the translations into the .strings subdirectory.
- Verify that all strings are correctly displayed in the application's GUI.

### 3.5.2 macOS GUI Utility Localization

All language files are located in the application package in the subdirectory "Contents/Resources". There is a separate subdirectory for each language. The name of this subdirectory consists of the identifier of the language and the extension ".lproj". This subdirectory contains the file with the strings of the language. The name of the file is always "Config.strings". For example, if the name of the application is

MyFirmwareUpdate.app

and one of the languages supported by the application is English (United States), the sub-path of the corresponding language file is

MyFirmwareUpdate.app/Contents/Resources/en-US.lproj/Config.strings

Each language-specific folder (such as en-US.lproj in the above example) may also contain an additional InfoPlist.strings file. This file contains certain language strings that are required to locate the services of macOS, such as the display of file information, the display of the application menu, and so on. The standard strings are taken from the Info.plist file of the application package (Contents/Info.plist, see section 3.4.2). If the system does not find an InfoPlist.strings file for the current system language, it uses the standard strings.

The application uses the current users preferred languages to select the language supported by the application. The user can define more than one preferred language in the system settings and assigns a priority to each language. If this is the case, the language selection algorithm described above is

slightly modified. The application at first tries to find a supported language for the preferred system language with the highest priority. If no match is found it tries the same with every the preferred system language with the next lower priority and so on. Only when it has found no supported language for all preferred system languages, it uses the built-in English text strings.

Note: The algorithm used by the system to select the appropriate `InfoPlist.strings` language file may differ slightly from the algorithm of the application described above.

To add a new language, follow these steps:

- Create a new language-specific folder under `Contents/Resources` by copying the original folder `en.lproj` including all files. Rename the folder depending on the desired language. For example, to add support for Chinese (Simplified), the new name of the new folder should be `zh-Hans.lproj`. It contains the files `Config.strings` and `InfoPlist.strings`.
- Send the new `Config.strings` file (which still contains the English strings) to a translator. When modifying the text file, the translator should follow the rules described in section 3.5.3.
- Optionally, send the file `InfoPlist.strings` to the translator as well.
- Replace the file `Config.strings` (and optionally `InfoPlist.strings`) in the new language folder after translation.
- Sign the application again, because the content of the package has changed.
- Verify that all strings are correctly displayed in the system's and application's GUI.

### 3.5.3 Language-specific Text Files Syntax

There are some rules to be followed when a language-specific file is created or modified to translate the strings:

- Language files use UTF-8 format. In macOS this is common. Windows often uses Unicode format instead. However, UTF-8 format is also possible. You can save a file in this format with Windows Notepad, for example. To do this, open the "File - Save As" dialog and select the appropriate encoding option.
- The file contains statements in the form

```
KEY = "String Value";
```

For example:

```
GROUP_UPGRADE_STATUSTEXT_IN_PROGRESS = "Upgrading firmware...";
```

- The text after the equals sign (=) may contain tokens in the form `${Identifier}` as placeholder for numbers or other data.

For example:

```
GROUP_UPGRADE_STATUSTEXT_TRANSFER_PROGRESS = "${CurrentBytes} of  
${TotalBytes} bytes transferred.";
```

- The file may also contain comments. A comment line begins with a double slash (//).

For example:

```
//This is a comment.
```

#### Instructions for a translator:

- The text in double quotation marks after the equals sign (=) must be translated.  
Example for a German translation:  

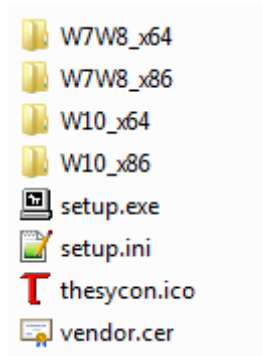
```
GROUP_UPGRADE_STATUSTEXT_IN_PROGRESS = "Firmware wird aktualisiert...";
```
- Do not translate comments, i.e. lines that begins with a double slash (//)!
- Do not modify any text before the equals sign (=).
- Do not modify or remove the quotation marks after the equals sign (=) and the semicolon at the end of the line.
- Do not translate or modify tokens in the form `${Identifier}`. These tokens will be replaced by values and must be contained in the translated text unmodified.
- Do not insert line breaks in the text to be translated. This changes the display of the text in the application. However, if the text editor automatically wraps the line at the end, this is not a problem as it is just the display in the editor.

## 4 Windows Driver Installer

ToriLogic provides a driver installer (setup.exe) which handles driver installation, uninstallation and update scenarios in a user-friendly and reliable way. An overview of the installer is given in the subsequent sections.

### 4.1 Driver Setup Files

The driver setup consists of various files arranged in a directory tree like this:



**Figure 2 Driver Setup Files Tree (top-level)**

If you ship these files to end users, the directory structure must be retained. Hence some kind of archive should be used. ToriLogic uses a self-extracting zip archive by default.

Note that the unpacked driver setup files are shipped as part of the DSK (see also 1.9.2).

### 4.2 Self-extracting Driver Install Package

ToriLogic provides a self-extracting setup package, for example:  
ToriLogic\_USBDFU\_v1.11.0\_2018-11-21\_setup.exe.

The self-extracting archive is created with the freeware utility 7-zip (<http://www.7-zip.org>).

This self-extracting archive is a delivery vehicle for the driver setup which consists of a bunch of files contained in a directory tree as described in the previous section. The self-extracting exe should not be confused with the driver installer (setup.exe) itself.

### 4.3 Driver Package Delivery to End Users

For delivery to end users, several options are available to licensees:

- 1) Deliver the self-extracting setup package that ToriLogic provides. This is the easiest way which requires no further work on your side.
- 2) Create another type of (self-extracting) archive using any tool of your choice to deliver the driver setup files. The directory structure shown above must be retained because setup.exe relies on it. See also 4.3.1 below.
- 3) Integrate the driver setup files into an overall software installer. This approach is recommended if you need to ship and install additional software components together with the driver. Refer to sections 4.4 and 4.5 for more information.

### 4.3.1 Why an exe should be Delivered Instead of a zip?

A zip file cannot be digitally signed. When a zip file is downloaded from the web then Windows possibly shows a big red warning message indicating that the file is potentially dangerous. This feature is called SmartScreen. For more information, check out

[https://en.wikipedia.org/wiki/Microsoft\\_SmartScreen](https://en.wikipedia.org/wiki/Microsoft_SmartScreen) for example.

Furthermore, when a downloaded zip is extracted then Windows marks each extracted file to indicate that it came from the web. If one of those files is launched then Windows shows a warning message.

These problems can be avoided by using a self-extracting exe archive which is digitally signed with an EV code signing certificate. Windows trusts this kind of archive because of its digital signature.

## 4.4 Integration with an Overall Software Installer

ToriLogic's driver installer (setup.exe) is not a general purpose software installer. If there are other considerable software components to be installed on the target system then an overall software installer should be created which includes the driver setup files as one component. A typical scenario is an application software package which includes the TL-USBDFU driver and possibly other driver packages to support devices used with that application.

To integrate the TL-USBDFU driver setup into your overall installer, you should:

- Include the directory tree with the driver setup files in your installer. Use the unpacked setup files which come with the DSK (see 4.1). Make sure the directory structure is retained.
- Have the installer place the driver setup files tree onto the target system, e.g. into a specific subdirectory of your main target directory.
- Run setup.exe in non-interactive (silent) mode. See the next section for more information.
- On uninstallation, also run setup.exe in non-interactive (silent) mode.

## 4.5 Driver Installer Silent Mode (Non-Interactive)

The driver installer setup.exe supports an interactive, GUI-based wizard-style mode which is the default. It can also be executed in silent mode (no GUI, no user interaction) by passing appropriate command line options. This is useful if the driver setup needs to be included as a component in an overall software installer which possibly installs many other components as well. Note that you should integrate the unpacked setup files, not the self-extracting setup archive. See also the previous section.

For details on how to use setup.exe in non-interactive mode, refer to the installer manual `pnpinstaller_manual.pdf` included with the DSK.



## 5 TL-USBDFU Software Development Kit (SDK)

To support integration of the TL-USBDFU driver with a custom application, ToriLogic provides a software development kit (SDK). The SDK comes with API header files, documentation and sample source code.

For detailed information on API functions and integration instructions, refer to the TL-USBDFU reference manual contained in the SDK. The reference manual is in HTML format. To open the documentation, run the script `start_refman.cmd` from the SDK's doc folder.

## 6 Analyzing Problems on Windows

In case the driver does not work as expected, there are some built-in diagnostic facilities available. This section provides information on how problems can be analyzed and gives some advice for specific situations.

### 6.1 Driver Installer Problem

If you experience problems with the driver installer (setup.exe) then follow the instructions below.

#### 6.1.1 Installer Log File

The driver installer (setup.exe) writes a log file which can be found in the following location:

```
%TEMP%\PnP Driver Installer.log
```

For detailed problem analysis, this log file should be emailed to ToriLogic.

Note that the path referred to by the environment variable %TEMP% is user-specific. If the user name is Tom, for example, and Windows is installed on drive C : then %TEMP% resolves to

```
C:\Users\Tom\AppData\Local\Temp
```

and thus the installer log file would be:

```
C:\Users\Tom\AppData\Local\Temp\PnP Driver Installer.log
```

The value of the %TEMP% variable can be queried by opening a console window (cmd.exe) and entering the command:

```
> set temp
```

#### 6.1.2 Setupapi Log Files

Log files generated by Windows during driver installation should also be sent to ToriLogic. These are all files that match the following pattern:

```
%SystemRoot%\inf\setupapi.*.log
```

Depending on the Windows version, this resolves to

```
C:\Windows\inf\setupapi.dev.log
```

```
C:\Windows\inf\setupapi.app.log
```

for example.

## **6.2 Error Code in Device Manager**

If Device Manager shows an error when you connect the USB device, there are several possible reasons for this as discussed in the following subsections.

### **6.2.1 Device Enumeration Failed**

If there are firmware issues and/or USB communication problems then Windows is not able to retrieve descriptors from the device. Consequently, Windows does not know the USB vendor ID (VID) and product ID (PID) and cannot load a driver for the device. On the device's property page in Device Manager you should check:

- Is a driver loaded for the device? Windows indicates this in the error message on the General tab and you can also check the Driver tab.
- Did Windows detect VID and PID correctly? You can verify this by checking Hardware Ids on the Details tab. If the hardware ID string does not correctly reflect VID and PID from your device's descriptors then there is a USB communication problem, or hardware/firmware issue.

### **6.2.2 Code Signing Issue**

If there is no error message from the driver then Windows was not able to load and execute the driver. Typically, this happens if there is a code-signing issue. Windows refuses to load a driver that is not correctly signed. If this happens on a Windows 7 system then the cause might be that the Windows update KB3033929 is not installed. See section 8.1 for more information.

## 7 Analyzing Problems on macOS

To capture debug output from the application, follow these steps:

- 1) Close the DFU application.
- 2) Open Finder. Run the Console application in the folder Applications/Utilities.
- 3) Click the leftmost button in the toolbar of the Console application to show the sidebar, if it's hidden.
- 4) Select your MAC in the group 'Devices' of the sidebar.
- 5) In the menu of the Console application:
  - Action:
    - Enable 'Include Info Messages'
    - Enable 'Include Debug Messages'
  - View -> Visible Columns:
    - Enable the following columns:  
Type, Date & Time, Process, PID, Thread ID, Subsystem, Message
- 6) In the search field of the Console application (top right):
  - Insert 'USB Device Firmware Upgrade'

Press Enter key. A selection list appears on the left edge of the search field. Select 'Process'.
- 7) Push the 'Clear' button of the Console application (top left) to empty the log view.
- 8) Start the DFU application and run it until the error occurs.
- 9) Select an arbitrary message in the message view of the Console application and press `<Command> + <A>` to select all messages.
- 10) Push the 'Share' button (top right) and select 'Mail' and send the logs to ToriLogic.  
Alternatively, you can also copy the messages with `<Command> + <C>`, save them to a file (e.g. via TextEdit) and send this file to ToriLogic.
- 11) In the toolbar (left) of the Console application select the 'system.log' entry and press `<Command> + <R>`. The finder will open and shows the location of the file system.log. Send this file also to ToriLogic.

## 8 Known Issues

### 8.1 Windows 7 requires hotfix to accept SHA-256 code signature

**Operating systems:**

Windows 7 32/64 bit

**Problem description:**

By default, Windows 7 does not accept SHA-256 code signatures, it accepts SHA-1 signatures only. However, SHA-1 based certificates are deprecated and not available any longer. All recent code signing certificates use the SHA-256 algorithm.

Microsoft has released a hotfix that adds support for SHA-256 certificates on Windows 7. The hotfix KB3033929 can be found here: <https://support.microsoft.com/en-us/kb/3033929>

Note that KB3033929 is available through Windows update. So the easiest solution is to make sure this update is installed.

**Additional information:**

This problem does not exist on Windows 8 and later.

If the driver installer (v1.12 and later) is used then setup.exe detects this problem during driver installation and shows an error message which references the KB3033929 patch.

### 8.2 Untrusted Publisher pop-up dialog appears on Windows 7

**Operating systems:**

Windows 7 32/64 bit

**Problem description:**

Normally, ToriLogic's driver installer suppresses the "Untrusted publisher" dialog box. However, if the driver that is installed is signed with a SHA-256 code signing certificate then Windows 7 might still show this dialog box. Although you selected the "Always trust" option, the dialog box might re-appear during a subsequent driver installation.

This is an issue in Windows 7. For more information and a hotfix, check out KB2921916:

<https://support.microsoft.com/en-us/kb/2921916>

**Additional information:**

KB2921916 is not available through Windows update.