	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b> A high-performance USB stack from Eclipse Foundation	<i>Document Version</i>	1.0
		<i>Release Date</i>	2024.07.08


# Safety Manual Eclipse USBX

## Version 6

### Revision 6.1.11

The high-performance USB stack from Eclipse Foundation

NOT FOR USE

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

## Disclaimer

This document is just a sample of the Functional Safety related documentation of the Eclipse USBX which is part of the Eclipse ThreadX Project.

This document is not intended for use.

This document and the associated software, while following the sample principles as the previously certified versions of the Eclipse ThreadX project, are not in any way certified.

## Copyright

Copyright ©2024 Eclipse Foundation AISBL. All Rights Reserved.

This document is the sole property of the Eclipse Foundation. Your use of this document is under the terms of the CC-BY-NC-ND 4.0 licence [[CC-BY-NC-ND-4.0](#)].

The Eclipse Foundation reserves the right to make changes to the specifications described herein at any time and without notice in order to improve design or reliability of Eclipse ThreadX.

The information in this document has been carefully checked for accuracy; however, the Eclipse Foundation makes no warranty pertaining to the correctness of this document.


## Licence

### CC-BY-NC-ND

This document is publicly shared under the CC BY-NC-ND 4.0 Deed Attribution-NonCommercial-NoDerivs 4.0 International [[CC-BY-NC-ND-4.0](#)] licence.


## Trademarks

ThreadX®, Eclipse ThreadX and the Eclipse ThreadX Logo are trademarks of the Eclipse Foundation AISBL.

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08


## Table of Contents

1. SCOPE .....	8
1.1. Purpose .....	8
1.2. Reference Standards .....	8
1.3. Safety related applications – requirements on safety lifecycle and the supporting processes .....	9
1.3.1. Safety related applications – requirements on safety lifecycle planning .....	9
1.3.2. Safety related applications – requirements on documentation .....	9
1.3.3. Safety related applications – requirements on configuration management .....	9
1.3.4. Safety related applications – requirements on change management .....	9
1.3.5. Safety related applications – risk management .....	9
1.4. Safety certifications .....	10
2. OVERVIEW OF USBX .....	10
2.1. USBX features .....	10
2.2. Products Highlights .....	11
2.3. Host, Device, OTG & Extensive Class Support .....	12
2.4. Powerful Services of USBX .....	14
2.4.1. Complete USB Device Framework Support .....	14
2.4.2. Easy-To-Use APIs .....	14
2.4.3. Multiple Host Controller Support .....	14
2.4.4. Software Scheduler .....	14
2.5. Required knowledge to integrate USBX .....	14
3. INSTALLATION OF USBX .....	15
3.1. Host Considerations .....	15
3.2. Target Considerations .....	15
3.3. Product Distribution .....	15
3.4. Configuration Options .....	16
3.5. Troubleshooting .....	18
3.6. USBX Version ID .....	18
4. FUNCTIONAL COMPONENTS OF THE USBX .....	18
4.1. Execution Overview of the USB Device Stack .....	18
4.1.1. Initialization .....	19
4.1.2. Application Interface Calls .....	19
4.1.3. USB Device Stack .....	19
4.1.4. USB Device Classes .....	20
4.1.5. VBUS Manager .....	23
4.1.6. Device Controller .....	23
4.1.7. USB Device Framework .....	24
4.2. Execution Overview of the USB Host Stack .....	25
4.2.1. Initialization .....	25
4.2.2. Application Interface Calls .....	26
4.2.3. USB Host Stack .....	26
4.2.4. USB Host Classes .....	29

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

4.2.5.	Host Controller .....	31
4.2.6.	USB Device Framework .....	32
5.	QUALIFICATION OF THE USBX SOFTWARE .....	34
<hr/>		
5.1.	Quality Management System .....	34
5.2.	USBX Quality Assurance .....	35
a)	Requirements .....	36
b)	Design .....	36
c)	Implementation .....	36
d)	Verification .....	37
e)	Maintenance .....	37
5.3.	Verification of the USBX Software .....	37
5.3.1.	Master Test Plan for the USBX 6.1.11 USB access system .....	37
5.3.2.	Test Code Coverage Analysis .....	38
5.3.3.	USBX Static Analysis .....	38
5.4.	Certification of USBX 6.1.11 .....	38
6.	CONCLUSION .....	39
<hr/>		
6.1.	Backward Compatibility .....	39
6.2.	Compatibility with other systems .....	39
6.3.	Requirements not met .....	39
6.4.	Outstanding Anomalies .....	39
6.5.	Design Safe State .....	39
6.6.	Functional Safety View .....	40
6.7.	Data Integrity .....	40
6.8.	Interface between the safety related application and USBX stack .....	40
APPENDICES .....		43
<hr/>		
Appendix [A]: TERMS AND ACRONYMS .....		43
[A1]	Reference Documents .....	43
[A2]	Acronyms .....	43


NOT FOR USE

 <b>ECLIPSE THREADX</b>	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

### LIST OF FIGURES


Figure 1: The relationship between the USB layers .....	11
Figure 2: Illustration of USBX Device Stack .....	19
Figure 3: Illustration of USBX Host stack .....	25
Figure 4: Shell window (excerpt) of the USBX 6.1.11 test suite .....	36
Figure 5: Usage of the USBX Software in a safety related context .....	41

NOT FOR USE




	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

This page is intentionally left blank


NOT FOR USE

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

## Guide conventions used

<i>Italics</i>	<i>typeface denotes book titles, emphasizes important words, and indicates variables</i>
<b>Bold</b>	emphasizes important words
	warning symbol draws attention to situations which could cause fatal errors
	information symbol draws attention to important or additional information
	references to detailed descriptions in USBX User Guide or the demonstration system

NOT FOR USE

	<b>Safety Manual</b>	<i>Document Status</i>	Released
		<i>Document Version</i>	1.0
	<b>USBX Version 6</b>	<i>Release Date</i>	2024.07.08
A high-performance USB stack from Eclipse Foundation			

## 1. SCOPE

### 1.1. Purpose

This Safety Manual provides comprehensive information about Eclipse USBX (hereafter called USBX), the high-performance USB foundation software from Eclipse Foundation.

This Safety Manual shall be read in context with the ThreadX User Guide [D8] explaining in detail the capabilities of this RTOS in view of its application in embedded systems and USBX User Guide [D1] giving information about the high-performance USB stack from Eclipse Foundation.

This manual provides guidance on following safety standards:

- IEC 61508 ([S1])
- IEC 62304 ([S2])
- ISO 26262 ([S3])
- EN 50128 ([S4])



*The USBX software version 6 (revision 6.1.11) was assessed by an accredited independent third-party certification body, for usability in development of safety related software.*

*Please consider that this certification is related only to unchanged USBX sources!*



*The USBX certification process by an accredited independent third party covers the generic USBX parts only*




*A manufacturer of a safety related application is highly recommended to follow the guidance of this Safety Manual and of the corresponding USBX Software documentation (e.g., User Guides).*

### 1.2. Reference Standards

Doc. No.	Standard Reference	Standard Title
[S1]	IEC 61508:2010	Functional safety of electrical/electronic/programmable electronic safety-related systems
[S2]	IEC 62304:2015	Medical device software - Software Life Cycle Processes
[S3]	ISO 26262:2018	Road vehicles - Functional safety
[S4]	EN 50128:2011	Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems

**Table 1: Reference Standards**



	<b>Safety Manual</b>	Document Status	Released
	<b>USBX Version 6</b>	Document Version	1.0
	A high-performance USB stack from Eclipse Foundation	Release Date	2024.07.08

### 1.3. Safety related applications – requirements on safety lifecycle and the supporting processes

#### 1.3.1. Safety related applications – requirements on safety lifecycle planning



*All safety related activities and procedures have to be planned for achieving functional safety. For that reason, all USBX related activities for a safety related application have to be considered in the overall safety plan.*

#### 1.3.2. Safety related applications – requirements on documentation

*Safety related applications which aim to fulfil the requirements specified in the functional safety standards [S1] - [S4] have to plan the documentation process in order to:*



- *Make the documentation available during each phase of the development processes*
- *Management of Functional Safety*
- *Make the documentation available for the functional safety assessment.*

*USBX documentation: Safety Manual, User Guide and Source Code Documentation are relevant for the product configuration and have to be considered in the documentation process.*

#### 1.3.3. Safety related applications – requirements on configuration management



*Safety related applications which aim to fulfil the requirements specified in the functional safety standards [S1] - [S4] have to be uniquely identified and reproduced in a controlled manner at any time. For that reason, the USBX configuration information contained in: Safety Manual, User Guide and Source Code Documentation shall be considered in customer's configuration management process.*

#### 1.3.4. Safety related applications – requirements on change management

*Change Management ensures planning, control, monitoring implementation and documentation of changes, while maintaining the consistency of each safety related application.*



*On that reason, all changes in the safety related application have to consider USBX anomalies and restrictions.*


*Furthermore, a re-base on a new USBX version during the safety lifecycle of a safety related application has to consider the USBX documentation related to the new version and to consider the USBX anomalies and restrictions in view of the application.*

#### 1.3.5. Safety related applications – risk management



*The safety related applications require hazard analysis and risk assessment or analysis of software contribution to hazardous situations.*

*USBX based safety related application has to consider in the hazard analysis and risk assessment or in the software risk management process all USBX restrictions and anomalies as specified in this Safety Manual, User Guide and Source Code Documentation.*

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

## 1.4. Safety certifications

The USBX software, version 6 (revision 6.1.11) has been assessed by an accredited independent third-party certification body (SGS-TÜV Saar GmbH) for usability in safety related systems, which have been developed according to following standards:

Safety Standard	Achieved Integrity Level
IEC 61508 [S1]	up to SIL 4
IEC 62304 [S2]	SW class C
ISO 26262 [S3]	up to ASIL D
EN 50128 [S4]	up to SW-SIL 4

For details to the certification refer to [D9].

## 2. OVERVIEW OF USBX

### 2.1. USBX features

USBX supports the three existing USB specifications: 1.1, 2.0 and OTG. It is designed to be scalable and will accommodate simple USB topologies with only one connected device as well as complex topologies with multiple devices and cascading hubs. USBX supports all the data transfer types of the USB protocols: control, bulk, interrupt, and isochronous.

USBX supports both the host side and the device side. Each side is comprised of three layers: •

- Controller layer
  - Stack layer
  - Class layer

The relationship between the USB layers is as follows:

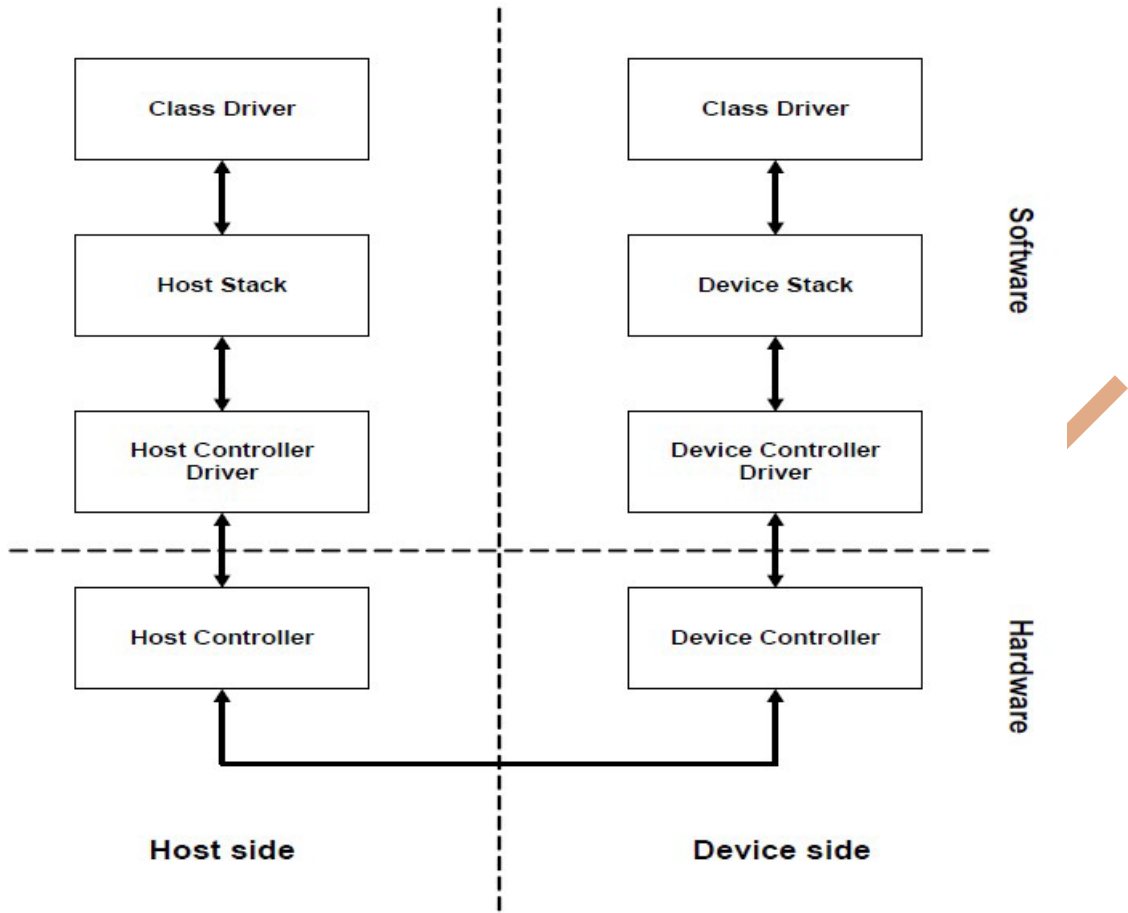



Figure 1: The relationship between the USB layers

## 2.2. Products Highlights

An overview of the USB stack's highlights follows:

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

USB Host Stack	USB Device Stack
<ul style="list-style-type: none"> <li>• Complete ThreadX processor support</li> <li>• No royalties</li> <li>• Complete ANSI C source code</li> <li>• Real-time performance</li> <li>• Responsive technical support</li> <li>• Multiple host controller support</li> <li>• Multiple class support</li> <li>• Multiple class instances</li> <li>• Integration of classes with ThreadX, FileX and NetX</li> <li>• Support for USB devices with multiple configuration</li> <li>• Support for USB composite devices</li> <li>• Support for cascading hubs</li> <li>• Support for USB power management</li> <li>• Support for USB OTG</li> <li>• Export trace events for TraceX</li> </ul>	<ul style="list-style-type: none"> <li>• Complete ThreadX processor support</li> <li>• No royalties</li> <li>• Complete ANSI C source code</li> <li>• Real-time performance</li> <li>• Responsive technical support</li> <li>• --</li> <li>• Multiple class support</li> <li>• Multiple class instances</li> <li>• Integration of classes with ThreadX, FileX and NetX</li> <li>• Support for USB devices with multiple configuration</li> <li>• Support for USB composite devices</li> <li>• --</li> <li>• Support for USB power management</li> <li>• Support for USB OTG</li> <li>• Export trace events for TraceX</li> </ul>

Eclipse USBX is a high-performance USB host, device, and on-the-go (OTG) embedded stack. USBX is fully integrated with Eclipse ThreadX and available for all Eclipse ThreadX-supported processors. Like ThreadX, USBX is designed to have a small footprint and high performance, making it ideal for deeply embedded applications that require an interface with USB devices.

### 2.3. Host, Device, OTG & Extensive Class Support


USBX Host/Device embedded USB protocol stack is an Industrial Grade embedded USB solution designed specifically for deeply embedded, real-time, and IoT applications. USBX provides host, device, and OTG support, as well as extensive class support. USBX is fully integrated with Eclipse ThreadX Real-Time Operating System, Eclipse FileX embedded FAT-compatible file system, Eclipse NetX, and Eclipse NetX Duo embedded TCP/IP stacks. All of this, combined with an extremely small footprint, fast execution and superior ease-of-use, make USBX the ideal choice for the most demanding embedded IoT applications requiring USB connectivity.

#### USBX memory footprint

USBX has a remarkably small minimal footprint of 10.5 KB of FLASH and 5.1 KB RAM for USBX Device CDC/ACM support. USBX Host requires a minimum of 18 KB of FLASH and 25 KB of RAM for CDC/ACM support.

An additional 10 KB to 13 KB of instruction area memory is needed for TCP functionality. USBX RAM usage typically ranges from 2.6 KB to 3.6 KB plus the packet pool memory, which is defined by the application.

Like ThreadX, the size of USBX automatically scales based on the services actually used by the application. This virtually eliminates the need for complicated configuration and build parameters, making things easier for the developer.

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

## USB Controller support

USBX supports major USB standards like OHCI and EHCI. In addition, USBX supports proprietary discrete USB host controllers from Atmel, Microchip, Philips, Renesas, ST, TI, and other vendors. USBX also supports multiple host controllers in the same application (i.e. running concurrently). USBX supports popular USB device controllers from Analog Devices, Atmel, Microchip, NXP, Philips, Renesas, ST, TI, and other vendors.

## Extensive Class support

USBX Host provides support for most popular classes, including ASIX, AUDIO, CDC/ACM, CDC/ECM, GSER, HID (keyboard, mouse, and remote control), HUB, PIMA (PTP/MTP), PRINTER, PROLIFIC, and STORAGE.

USBX Device provides support for most popular classes, including CDC/ACM, CDC/ECM, DFU, HID, PIMA (PTP/MTP) (w/MTP), RNDIS, and STORAGE. Support for custom classes is also available.

## Pictbridge support

USBX supports the full Pictbridge implementation both on the host and the device. Pictbridge sits on top of USBX PIMA (PTP/MTP) class on both sides. The PictBridge standards allow the connection of a digital still camera or a smart phone directly to a printer without a PC, enabling direct printing to certain Pictbridge aware printers.

When a camera or phone is connected to a printer, the printer is the USB host and the camera is the USB device. However, with Pictbridge, the camera will appear as being the host and commands are driven from the camera. The camera is the storage server, the printer the storage client. The camera is the print client and the printer is of course the print server. Pictbridge uses USB as a transport layer but relies on PTP (Picture Transfer Protocol) for the communication protocol.


## Custom Class support

USBX Host and Device support custom classes. An example custom class is provided in the USBX distribution. This simple data pump class is called DPUMP and can be used as a model for custom application classes. Advanced technology USBX Host and Device support custom classes. An example custom class is provided in the USBX distribution. USBX is advanced technology that includes:

- Host, Device, and OTG support
- USB low, full, and high-speed support
- Automatic scaling
- Fully integrated with Eclipse ThreadX, Eclipse FileX, and Eclipse NetX
- Optional performance metrics
- Eclipse TraceX system analysis support

USBX supports OTG in the core USB stack. But for OTG to function, it requires a specific USB controller. USBX OTG controller functions can be found in the `usbx_otg` directory. The current USBX version only supports the NXP LPC3131 with full OTG capabilities.

The regular controller driver functions (host or device) can still be found in the standard USBX `usbx_device_controllers` and `usbx_host_controllers` but the `usbx_otg` directory contains the specific OTG functions associated with the USB controller.

	<b>Safety Manual</b>	<small>Document Status</small>	Released
	<b>USBX Version 6</b>	<small>Document Version</small>	1.0
	A high-performance USB stack from Eclipse Foundation	<small>Release Date</small>	2024.07.08

There are four categories of functions for an OTG controller in addition to the usual host/device functions.

- VBUS specific functions: Each controller needs to have a VBUS manager to change the state of VBUS based on power management requirements. Usually, this function only performs turning on or off VBUS.
- Start and Stop of the controller: Unlike a regular USB implementation, OTG requires the host and/or the device stack to be activated and deactivated when the role changes.
- USB role manager: The USB role manager receives commands to change the state of the USB.



*For more details to the states that need transitions to and from refer to section (USB role manager in chapter 5 of the USB User Guide (Device Stack) [D1]).*

- Interrupt handlers: Both host and device controller drivers for OTG needs different interrupt handlers to monitor signals beyond traditional USB interrupts, in particular signals due to SRP and VBUS.

## 2.4. Powerful Services of USBX

### 2.4.1. Complete USB Device Framework Support

USBX can support the most demanding USB devices, including multiple configurations, multiple interfaces, and multiple alternate settings.

### 2.4.2. Easy-To-Use APIs

USBX provides the very best deeply embedded USB stack in a manner that is easy to understand and use. The USBX API makes the services intuitive and consistent. By using the provided USBX class APIs, the user application does not need to understand the complexity of the USB protocols.

### 2.4.3. Multiple Host Controller Support

USBX can support multiple USB host controllers running concurrently. This feature allows USBX to support the USB 2.0 standard using the backward compatibility scheme associated with most USB 2.0 host controllers on the market today.


### 2.4.4. Software Scheduler

USBX host contains a USB software scheduler necessary to support USB controllers that do not have hardware list processing. The USBX software scheduler will organize USB transfers with the correct frequency of service and priority and will instruct the USB controller to execute each transfer.

## 2.5. Required knowledge to integrate USBX

USBX is intended for the embedded real-time software developer. The developer should be familiar with standard real-time operating system functions, the USB specification, and the C programming language.



	<b>Safety Manual</b>	<small>Document Status</small>	<small>Released</small>
	<b>USBX Version 6</b>	<small>Document Version</small>	1.0
	<small>A high-performance USB stack from Eclipse Foundation</small>	<small>Release Date</small>	2024.07.08



For technical information related to USB, see the USB specification and USB Class specifications that can be downloaded at <https://www.USB.org/developers>

### 3. INSTALLATION OF USBX

#### 3.1. Host Considerations

Embedded development is usually performed on Windows PC or Unix host computers. After the application is compiled, linked, and the executable is generated on the host, it is downloaded to the target hardware for execution.

Usually, the target download is done over an RS-232 serial interface, although parallel interfaces, USB, and Ethernet are becoming more popular. See the development tool documentation for available options.

Debugging is done typically over the same link as the program image download. A variety of debuggers exist, ranging from small monitor programs running on the target through Background Debug Monitor (BDM) and In-Circuit Emulator (ICE) tools. Of course, the ICE tool provides the most robust debugging of actual target hardware.

As for resources used on the host, the source code for USBX is delivered in ASCII format and requires approximately 500 Kbytes of space on the host computer's hard disk.



Review the supplied `readme_usbx_generic.txt` file for additional host system considerations and options

#### 3.2. Target Considerations


USBX requires between 24 KBytes and 64 KBytes of Read Only Memory (ROM) on the target in host mode. The amount of memory required is dependent on the type of controller used and the USB classes linked to USBX. Another 32 KBytes of the target's Random Access Memory (RAM) are required for USBX global data structures and memory pool. This memory pool can also be adjusted depending on the expected number of devices on the USB and the type of USB controller. The USBX device side requires roughly 10-12 KBytes of ROM depending on the type of device controller. The RAM memory usage depends on the type of class emulated by the device. USBX also relies on ThreadX semaphores, mutexes, and threads for multiple thread protection, and I/O suspension and periodic processing for monitoring the USB bus topology.

#### 3.3. Product Distribution

USBX can be obtained from our public source code repository at <https://github.com/eclipse-threadx/usbx>.

Following is a list of the important files common to most product distributions:

<b><code>readme_usbx.txt</code></b>	<i>This file contains specific information about the USBX port, including information about the target processor and the development tools.</i>
<b><code>ux_api.h</code></b>	<i>This C header file contains all system equates, data structures, and service prototypes.</i>

	<b>Safety Manual</b>	Document Status	Released
	<b>USBX Version 6</b>	Document Version	1.0
	A high-performance USB stack from Eclipse Foundation	Release Date	2024.07.08

<b><i>ux_port.h</i></b>	<i>This C header file contains all development-tool--specific data definitions and structures.</i>
<b><i>ux.lib</i></b>	<i>This is the binary version of the USBX C library. It is distributed with the standard package.</i>
<b><i>demo_usbxc.c</i></b>	<i>The C file containing a simple USBX demo</i>



All files are in lower-case, making it easy to convert the commands to Linux (Unix) development platforms.

USBX is installed by cloning the GitHub repository to your local machine. The following is typical syntax for creating a clone of the USBX repository on your PC:

```
git clone https://github.com/eclipse-threadx/usbxc.git
```

Alternatively, you can download a copy of the repository using the download button on the GitHub main page.

The following instructions apply virtually to any installation:


Step 1:	<i>Backup the USBX distribution disk and store it in a safe location.</i>
Step 2:	<i>Use the same directory in which you previously installed ThreadX on the host hard drive. All USBX names are unique and will not interfere with the previous USBX installation.</i>
Step 3:	<i>Add a call to <b>ux_system_initialize</b> at or near the beginning of <b>tx_application_define</b>. This is where the USBX resources are initialized.</i>
Step 4:	<i>Add a call to <b>ux_host_stack_initialize</b>.</i>
Step 5:	<i>Add one or more calls to initialize the required USBX</i>
Step 6:	<i>Add one or more calls to initialize the host controllers available in the system</i>
Step 7:	<i>It may be required to modify the <b>tx_low_level_initialize.c</b> file to add low level hardware initialization and interrupt vector routing. This is specific to the hardware platform and will not be discussed here.</i>
Step 8:	<i>Compile application source code and link with the USBX and ThreadX run time libraries (FileX and/or Netx may also be required if the USB storage class and/or USB network classes are to be compiled in), <b>ux.a</b> (or <b>ux.lib</b>) and <b>tx.a</b> (or <b>tx.lib</b>). The resulting can be downloaded to the target and executed!</i>

### 3.4. Configuration Options


There are several configuration options for building the USBX library and these are located in the **ux\_user.h**.

Following are some examples of configuration options:



	<b>Safety Manual</b>		Document Status	Released
	<b>USBX Version 6</b>		Document Version	1.0
	A high-performance USB stack from Eclipse Foundation		Release Date	2024.07.08

Host Stack	Device Stack
<p><b><u>UX_PERIODIC_RATE</u></b></p> <p>This value represents how many ticks per seconds for a specific hardware platform. The default is 1000 indicating 1 tick per millisecond.</p>	<p><b><u>UX_MAX_SLAVE_CLASS_DRIVER</u></b></p> <p>This is the maximum number of USBX classes that can be registered via <b><u>ux_device_stack_class_register</u></b>.</p>
<p><b><u>UX_MAX_HCD</u></b></p> <p>This value represents the number of different host controllers that are available in the system. For USB 1.1 support, this value will mostly be 1. For USB 2.0 support this value can be more than 1. This value represents the number of concurrent host controllers running at the same time. If for instance, there are two instances of OHCI running or one EHCI and one OHCI controllers running, the <b><u>UX_MAX_HCD</u></b> should be set to 2.</p>	<p><b><u>UX_THREAD_PRIORITY_DCD</u></b></p> <p>This is the ThreadX priority value for the device controller thread.</p>
<p><b><u>UX_MAX_DEVICES</u></b></p> <p>This value represents the maximum number of devices that can be attached to the USB. Normally, the theoretical maximum number on a single USB is 127 devices. This value can be scaled down to conserve memory. It should be noted that this value represents the total number of devices regardless of the number of USB buses in the system.</p>	<p><b><u>UX_NO_TIME_SLICE</u></b></p> <p>Defined, the ThreadX target port does not use time slices.</p>
<p><b><u>UX_MAX_ED</u></b></p> <p>This value represents the maximum number of EDs in the controller pool. This number is assigned to one controller only. If multiple instances of controllers are present, this value is used by each individual controller.</p>	<p><b><u>UX_SLAVE_REQUEST_DATA_MAX_LENGTH</u></b></p> <p>This value represents the maximum number of bytes received on a bulk endpoint in the device stack. The default is 4096 bytes but can be reduced in memory constraint environments.</p>
<p><b><u>UX_THREAD_STACK_SIZE</u></b></p> <p>This value is the size of the stack in bytes for the USBX threads. It can be typically 1024 or 2048 bytes depending on the processor used and the host controller.</p>	
<p><b><u>UX_MAX_HOST_LUN</u></b></p> <p>This value represents the maximum number of SCSI logical units represented in the host storage class driver.</p>	<p><b><u>UX_MAX_SLAVE_LUN</u></b></p> <p>This value represents the current number of SCSI logical units represented in the device storage class driver.</p>

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08



The complete lists of the configuration options are described in section “Configuration options” of chapter 3 of the USBX User Guide



Additional development tool options are described in the **readme\_usbx.txt** file supplied on the distribution disk

### 3.5. Troubleshooting

USBX is delivered with a demonstration file and a simulation environment. It is always a good idea to get the demonstration platform running first—either on the target hardware or a specific demonstration platform.

In addition, it’s also very useful to examine the traffic on the USB bus using a USB Protocol Analyzer. This information is also very useful to provide when contacting support.

If you encounter any bugs, have suggestions for new features, or if you would like to become an active contributor to this project, please follow the instructions provided in the contribution guideline for the corresponding repo.

### 3.6. USBX Version ID

The current version of USBX is available to both the user and the application software during runtime.

The programmer can obtain the USBX version from examination of the **ux\_port.h** file. In addition, this file also contains a version history of the corresponding port. Application software can obtain the USBX version by examining the global string **\_ux\_version\_id**, which is defined in **ux\_port.h**.

## 4. FUNCTIONAL COMPONENTS OF THE USBX

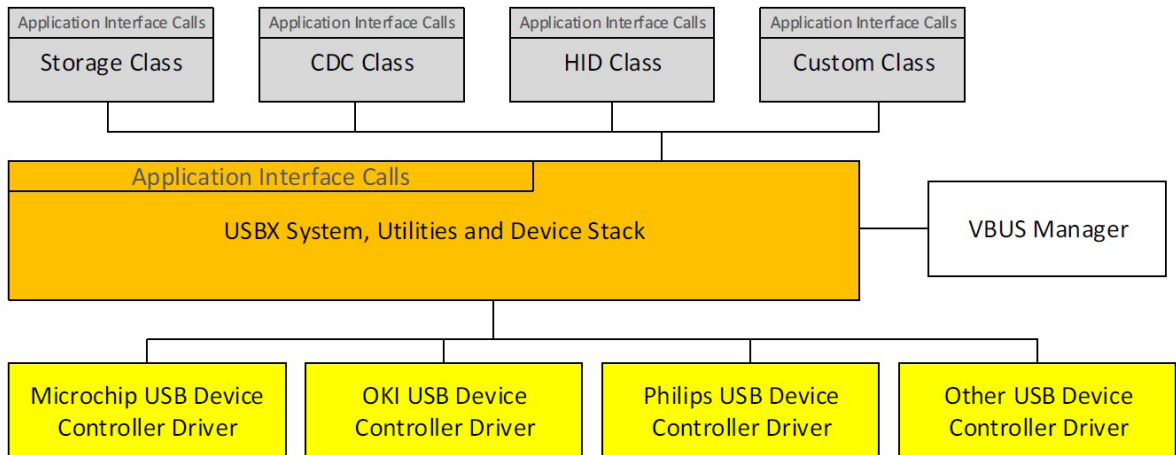
This chapter includes a description of the high performance USBX embedded USB host stack and device stack from a functional perspective.

### 4.1. Execution Overview of the USB Device Stack

USBX for the device is composed of several components:

- Initialization
- USB Device Classes
- USB Device Stack
- Device Controller
- VBUS manager
- Application Interface Calls

Following figure illustrates the USBX Device stack:



**Figure 2: Illustration of USBX Device Stack**  
(Source: USBX User Guide (Device Stack) [D1])

#### 4.1.1. Initialization

In order to activate USBX, the function `ux_system_initialize` must be called. This function initializes the memory resources of USBX.

In order to activate USBX device facilities, the function `ux_device_stack_initialize` must be called. This function will in turn initialize all the resources used by the USBX device stack such as ThreadX threads, mutexes, and semaphores.

It is up to the application initialization to activate the USB device controller and one or more USB classes. Contrary to the USB host side, the device side can have only one USB controller driver running at any time. When the classes have been registered to the stack and the device controller(s) initialization function has been called, the bus is active and the stack will reply to bus reset and host enumeration commands.

#### 4.1.2. Application Interface Calls


There are two levels of APIs in USBX:

- USB Device Stack APIs (refer to section 4.1.3), and
- USB Device Class APIs (refer to section 4.1.4),

Normally, an USBX application should not have to call any of the USB Host Stack APIs. Most applications will only access the USB Host Classes APIs.

#### 4.1.3. USB Device Stack

The device stack APIs are responsible for the registration of USBX device components such as classes and the device framework.

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

Following table lists a subset of the USBX Device Stack services and their descriptions:

Prototype	Description
UINT <b>ux_device_stack_alternate_setting_get</b> (ULONG interface_value)	This function is used by the USB host to obtain the current alternate setting for a specific interface value. It is called by the controller driver when a GET_INTERFACE request is received.  <b>Return Values</b> <b>UX_SUCCESS</b> (0x00) The data transfer was completed. <b>UX_ERROR</b> (0xFF) Wrong interface value.
UINT <b>ux_device_stack_configuration_get</b> (VOID)	This function is used by the host to obtain the current configuration running in the device.  <b>Return Values</b> <b>UX_SUCCESS</b> (0x00) The data transfer was completed.
UINT <b>ux_device_stack_endpoint_stall</b> (UX_SLAVE_ENDPOINT *endpoint)	This function is called by the USB device class when an endpoint should return a Stall condition to the host.  <b>Return Values</b> <b>UX_SUCCESS</b> (0x00) The operation was successful. <b>UX_ERROR</b> (0xFF) The device is in an invalid state.



The complete list of the USBX Device Services and their descriptions can be found in chapter 4 of the USB User Guide (Device Stack) [D1].

#### 4.1.4. USB Device Classes


The Device Class APIs are very specific to each USB class. Most of the common APIs for USB classes provided services such as opening/closing a device and reading from and writing to a device.

##### Storage Class

The storage class is in charge of answering storage class specific control requests and handling storage class protocol commands.

The USB device storage class allows for a storage device embedded in the system to be made visible to a USB host.

The USB device storage class does not by itself provide a storage solution. It merely accepts and interprets SCSI requests coming from the host. When one of these requests is a read or a write

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
A high-performance USB stack from Eclipse Foundation		<i>Release Date</i>	2024.07.08

command, it will invoke a pre-defined call back to a real storage device handler, such as an ATA device driver or a Flash device driver.




Refer to Storage part in Chapter 5 for more information


### CDC Class

The CDC class is in charge of answering CDC class specific control requests and offering ways to communicate with host through data pipes. Following functionalities are supported now:

- CDC-ACM (or CDC/ACM): communicate with host as a serial device

The USB device CDC-ACM class allows for a USB host system to communicate with the device as a serial device. This class is based on the USB standard and is a subset of the CDC standard. Following table lists a subset of the USBX Device CDC-ACM Class APIs:

Prototype	Description
UINT <b>ux_device_class_cdc_acm_ioctl</b> ( UX_SLAVE_CLASS_CDC_ACM *cdc_acm, ULONG ioctl_function, VOID *parameter)	This function is called when an application needs to perform various ioctl calls to the cdc acm interface.  <b>Return Values</b> <b>UX_SUCCESS</b> (0x00) This operation was successful. <b>UX_Error</b> (0X00) Error from function.
UINT <b>ux_device_class_cdc_acm_read</b> (UX_SLAVE_CLASS_CDC_ACM *cdc_acm, UCHAR *buffer, ULONG requested_length, ULONG *actual_length)	This function is called when an application needs to read from the OUT data pipe (OUT from the host, IN from the device). It is blocking.  <b>Return Values</b> <b>UX_SUCCESS</b> (0x00) This operation was successful. <b>UX_CONFIGURATION_HANDLE_UNKNOWN</b> (0x51) Device is no longer in the configured state  <b>UX_TRANSFER_NO_ANSWER</b> (0X22) No answer from device. The device was probably disconnected while the transfer was pending.  <b>UX_TRANSFER_BUFFER_OVERFLOW</b> (0X27) Transfer buffer overflow, inside a USB packet, host sending more bytes than available buffer.   <i>Note the function reads bytes from the host packet by packet. If the prepared buffer size is smaller than a packet and the host sends more data than expected (in other words, the prepared buffer size is not a multiple of the USB endpoint's max packet size), then buffer overflow will occur. To avoid this issue, the recommended way to read is to allocate a buffer exactly one packet size (USB endpoint max packet size). This way if there is more data, the next</i>

	<b>Safety Manual</b>	Document Status	Released
	<b>USBX Version 6</b>	Document Version	1.0
A high-performance USB stack from Eclipse Foundation		Release Date	2024.07.08

	<p><i>read can get it and no buffer overflow will occur. If there is less data, the current read can get a short packet instead of generating an error.</i></p>
<pre> UINT ux_device_class_cdc_acm_write_with _callback(UX_SLAVE_CLASS_CDC_ACM *cdc_acm, UCHAR *buffer, ULONG requested_length) </pre>	<p>This function is called when an application needs to write to the IN data pipe (IN from the host, OUT from the device). This function is non-blocking and the completion will be done through a callback.</p> <p><b>Return Values</b></p> <p><b>UX_SUCCESS</b> (0x00) This operation was successful.</p> <p><b>UX_CONFIGURATION_HANDLE_UNKNOWN</b> (0x51) Device is no longer in the configured state</p> <p><b>UX_TRANSFER_NO_ANSWER</b> (0x22) No answer from device. The device was probably disconnected while the transfer was pending.</p>



Refer to chapter 5 of the “USBX User Guide” [D1] for the complete list of the CDC-ACM Class APIs and their descriptions.

- CDC/ECM (or CDC-ECM): communicate with host as an ethernet device

The USB device CDC-ECM class allows for a USB host system to communicate with the device as an ethernet device. This class is based on the USB standard and is a subset of the CDC standard.



Refer to chapter 5 of the “USBX User Guide” [D1] for more details.

### HID Class

The HID class is in charge of answering the HID class specific control requests and offering ways to communicate host with HID class specific reports.

USBX HID device class is relatively simple compared to the host side. It is closely tied to the behavior of the device and its HID descriptor.




Refer to HID part in Chapter “USBX User guide” [D1] for more information

### Custom Class

For advanced developers, it’s possible to create more customized class, to answering customized control requests and handling customized protocol on data pipes. Note such class may also require specific customization on host side, too.



	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
A high-performance USB stack from Eclipse Foundation		<i>Release Date</i>	2024.07.08

## Storage Class

The storage class is in charge of answering storage class specific control requests and handling storage class protocol commands.

The USB device storage class allows for a storage device embedded in the system to be made visible to a USB host. The USB device storage class does not by itself provide a storage solution. It merely accepts and interprets SCSI requests coming from the host.

When one of these requests is a read or a write command, it will invoke a pre-defined call back to a real storage device handler, such as an ATA device driver or a Flash device driver.



Refer to Storage part in Chapter 5 of the “USBX User guide” [D1] for more information

### 4.1.5. VBUS Manager

In most USB device designs, VBUS is not part of the USB Device core but rather connected to an external GPIO, which monitors the line signal.

As a result, VBUS has to be managed separately from the device controller driver.

It is up to the application to provide the device controller with the address of the VBUS IO. VBUS must be initialized prior to the device controller initialization.

Depending on the platform specification for monitoring VBUS, it is possible to let the controller driver handle VBUS signals after the VBUS IO is initialized or if this is not possible, the application has to provide the code for handling VBUS.


If the application wishes to handle VBUS by itself, its only requirement is to call the function ***ux\_device\_stack\_disconnect()*** when it detects that a device has been extracted. It is not necessary to inform the controller when a device is inserted because the controller will wake up when the BUS RESET assert/deassert signal is detected.

### 4.1.6. Device Controller

The device controller driver (DCD) interoperates USB Device Stack operations to hardware actions. Normally, a USBX application should not have to call device controller APIs, except initialization function. When the device controller initialization function is called, the bus is active and the stack will reply to bus reset and host enumeration commands through device controller driver.

Here are some possible hardware which USB Device Stack can operate on:

- Microchip chip with USB device controller
- OKI chip with USB device controller
- Philips chip with USB device controller
- Other chip with USB device controller, etc.

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
A high-performance USB stack from Eclipse Foundation		<i>Release Date</i>	2024.07.08

#### 4.1.7. USB Device Framework

The USB device side is responsible for the definition of the device framework. The device framework is divided into three categories:

- Definition of the components of the Device Framework: The definition of each component of the device framework is related to the nature of the device and the resources utilized by the device. Following are the main categories.
  - Device Descriptor
  - Configuration Descriptor
  - Interface Descriptor
  - Endpoint Descriptor

USBX supports device component definition for both high and full speed (low speed being treated the same way as full speed). This allows the device to operate differently when connected to a high speed or full speed host. The typical differences are the size of each endpoint and the power consumed by the device. The definition of the device component takes the form of a byte string that follows the USB specification. The definition is contiguous and the order in which the framework is represented in memory will be the same as the one returned to the host during enumeration.



Refer to section “Definition of the components of the device framework” in chapter 3 of the USB User Guide [D1] for an example of a device framework for a high speed Flash Disk.

- Definition of the Strings of the Device Framework: Strings are optional in a device. Their purpose is to let the USB host know about the manufacturer of the device, the product name, and the revision number through Unicode strings. The main strings are indexes embedded in the device descriptors. Additional strings indexes can be embedded into individual interfaces. For more details refer to section “Definition of the strings of the device framework” in chapter 3 of the USB User Guide [D2].



Refer to section “Definition of the Strings of the device framework” in chapter 3 of the USB User Guide [D1] for an example of a device framework for a high-speed Flash Disk.

- Definition of the Languages Supported by the Device Framework: USBX has the ability to support multiple languages although English is the default. The definition of each language for the string descriptors is in the form of an array of languages definition defined as follows:

```
#define LANGUAGE_ID_FRAMEWORK_LENGTH 2
  UCHAR language_id_framework[] = {
    /* English. */
    0x09, 0x04
  };
```

To support additional languages, simply add the language code double-byte definition after the default English code.



The language code has been defined by Microsoft in the document: *Developing International Software for Windows 95 and Windows NT*, Nadine Kano, Microsoft Press, Redmond WA.

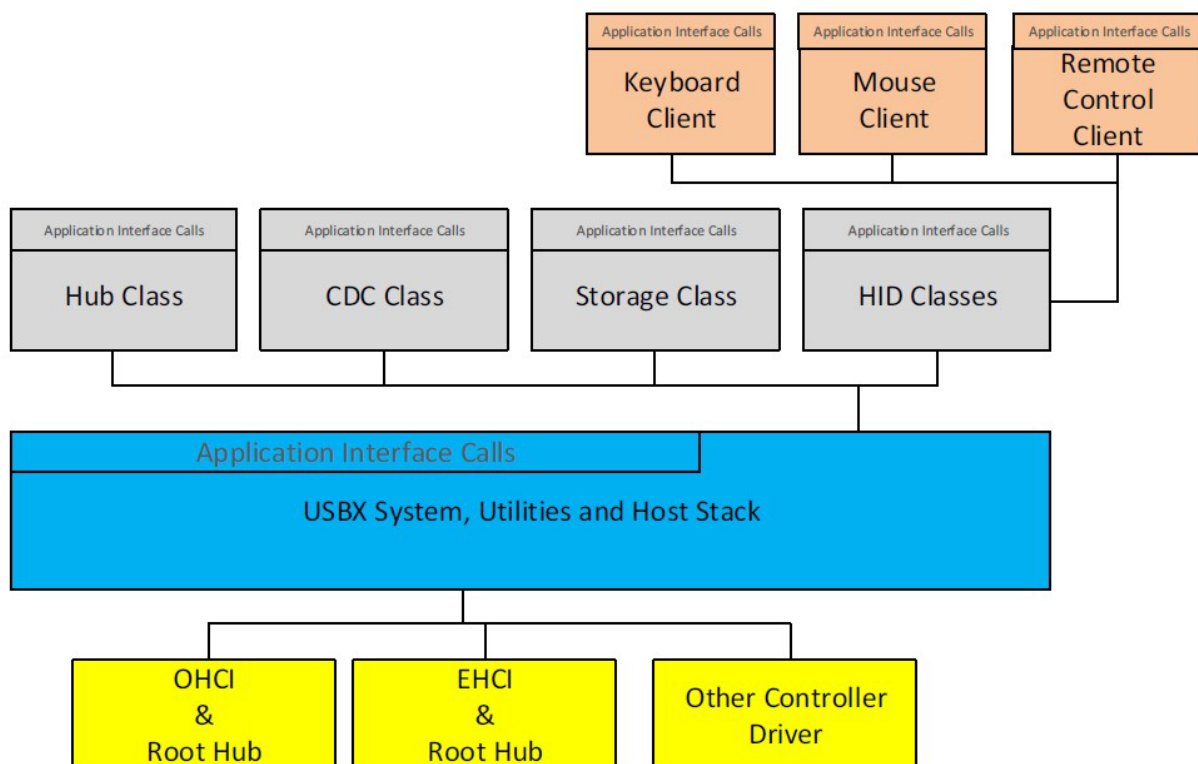


## 4.2. Execution Overview of the USB Host Stack

USBX is composed of several components:

- Initialization
- Application interface calls
- Root Hub
- Host Classes
- Hub Class
- USB Host Stack
- Host Controller

The following diagram illustrates the USBX host stack.




**Figure 3: Illustration of USBX Host stack**

(Source: USBX Host Stack User Guide [D1])

### 4.2.1. Initialization

In order to activate USBX, the function ***ux\_system\_initialize*** must be called. This function initializes the memory resources of USBX.

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

In order to activate USBX device facilities, the function ***ux\_host\_stack\_initialize*** must be called. This function will in turn initialize all the resources used by the USBX device stack such as ThreadX threads, mutexes, and semaphores.

It is up to the application initialization to activate at least one USB host controller and one or more USB classes. When the classes have been registered to the stack and the host controller(s) initialization function has been called the bus is active and device discovery can start. If the root hub of the host controller detects an attached device, the USB enumeration thread, in charge of the USB topology, will be wake up and proceed to enumerate the device(s).

It is possible, due to the nature of the root hub and downstream hubs, that all attached USB devices may not have been configured completely when the host controller initialization function returns. It can take several seconds to enumerate all USB devices, especially if there are one or more hubs between the root hub and USB devices.

#### 4.2.2. Application Interface Calls

here are two levels of APIs in USBX:

- USB Host Stack APIs (refer to section 4.2.3), and
- USB Host Class APIs (refer to section 4.2.4)

Normally, a USBX application should not have to call any of the USB host stack APIs. Most applications will only access the USB Class APIs.

#### 4.2.3. USB Host Stack

The USB host stack is the centrepiece of USBX und has three main functions:


- Manage the topology of the USB: The USB stack topology thread is awakened when a new device is connected or when a device has been disconnected. Either the root hub or a regular hub can accept device connections. Once a device has been connected to the USB, the topology manager will retrieve the device descriptor. This descriptor will contain the number of possible configurations available for this device. Most devices have one configuration only. Some devices can operate differently according to the available power available on the port where it is connected. If this is the case, the device will have multiple configurations that can be selected depending on the available power. When the device is configured by the topology manager, it is then allowed to draw the amount of power specified in its configuration descriptor.
- Bind a USB device to one or more classes: When the device is configured, the topology manager will let the class manager continue the device discovery by looking at the device interface descriptors. A device can have one or more interface descriptors.

An interface represents a function in a device. For instance, a USB speaker has three interfaces, one for audio streaming, one for audio control, and one to manage the various speaker buttons.

The class manager has two mechanisms to join the device interface(s) to one or more classes. It can either use the combination of a PID/VID (product ID and vendor ID) found in the interface descriptor or the combination of Class/Subclass/Protocol.

The PID/VID combination is valid for interfaces that cannot be driven by a generic class. The Class/Subclass/Protocol combination is used by interfaces that belong to a USB-IF certified class such as a printer, hub, storage, audio, or HID.

The class manager contains a list of registered classes from the initialization of USBX. The class manager will call each class one at a time until one class accepts to manage the interface

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
A high-performance USB stack from Eclipse Foundation		<i>Release Date</i>	2024.07.08


for that device. A class can only manage one interface. For the example of the USB audio speaker, the class manager will call all the classes for each of the interfaces. Once a class accepts an interface, a new instance of that class is created. The class manager will then search for the default alternate setting for the interface. A device may have one or more alternate settings for each interface. The alternate setting 0 will be the one used by default until a class decides to change it. For the default alternate setting, the class manager will mount all the endpoints contained in the alternate setting. If the mounting of each endpoint is successful, the class manager will complete its job by returning to the class that will finish the initialization of the interface.

- Exports a certain number of APIs for the USB classes to perform interrogation on the device and USB transfers on specific endpoints. APIs are described in detail in the USBX Host Device Guide [D1].


The host stack APIs are responsible for the registration of USBX components (host classes and host controllers), configuration of devices, and the transfer requests for available device endpoints.

Following table lists a subset of the USBX Host Stack APIs and their descriptions:

Prototype	Description
<pre> UINT ux_host_stack_initialize(UINT (*system_change_function) (ULONG, UX_HOST_CLASS *)) </pre>	<p>This function will initialize the USB host stack. The supplied memory area will be setup for USBX internal use. If <code>UX_SUCCESS</code> is returned, USBX is ready for host controller and class registration.</p> <p><b>Return Values:</b>  <b>UX_SUCCESS</b> (0x00) Successful initialization.  <b>UX_MEMORY_INSUFFICIENT</b> (0x12) A memory allocation failed.</p>
<pre> UINT ux_host_stack_endpoint_transfer_abort (UX_ENDPOINT *endpoint) </pre>	<p>This function will cancel all transactions active or pending for a specific transfer request attached to an endpoint and return immediately. If the transfer request has a callback function attached; the callback function will be called with the <code>UX_TRANSACTION_ABORTED</code> status.</p> <p><b>Return Values</b>  <b>UX_SUCCESS</b> (0x00) No errors.  <b>UX_ENDPOINT_HANDLE_UNKNOWN</b> (0x53) Endpoint handle is not valid</p>

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
A high-performance USB stack from Eclipse Foundation		<i>Release Date</i>	2024.07.08

<pre> UINT ux_host_stack_class_get(UCHAR *class_name, UX_HOST_CLASS **class) </pre>	<p>This function returns a pointer to the class container immediately. A class needs to obtain its container from the USB stack to search for instances when a class or an application wants to open a device.</p> <p><b>Return Values</b></p> <p><b>UX_SUCCESS</b> (0x00) No errors, on return the class field is field with the pointer to the class container</p> <p><b>UX_HOST_CLASS_UNKNOWN</b> (0X59) Class is unknown by the stack</p>
<pre> UINT ux_host_stack_class_instance_create(UX_H OST_CLASS *class,VOID *class_instance) </pre>	<p>This function creates a new class instance for a class container and returns immediately. The instance of a class is not contained in the class code to reduce the class complexity. Rather, each class instance is attached to the class container located in the main stack.</p> <p><b>Return Values</b></p> <p><b>UX_SUCCESS</b> (0x00) The class instance was attached to the class container.</p>
<pre> UINT ux_host_stack_class_instance_destroy(UX_ HOST_CLASS *class, VOID *class_instance) </pre>	<p>This function destroys a class instance for a class container. <b>Return Values:</b></p> <p><b>UX_SUCCESS</b> (0x00) The class instance was destroyed.</p> <p><b>UX_HOST_CLASS_INSTANCE_UNKNOWN</b> (0x5b) The class instance is not attached to the class container.</p>
<pre> UINT ux_host_stack_interface_endpoint_get(UX_ INTERFACE *interface, UINT endpoint_index, UX_ENDPOINT **endpoint) </pre>	<p>This function returns an endpoint container based on the interface handle and an endpoint index. It is assumed that the alternate setting for the interface has been selected or the default setting is being used prior to the endpoint(s) being searched. <b>Return Values:</b></p> <p><b>UX_SUCCESS</b> (0x00) The endpoint container exists and is returned.</p> <p><b>UX_INTERFACE_HANDLE_UNKNOWN</b> (0x52) Interface specified does not exist.</p> <p><b>UX_ENDPOINT_HANDLE_UNKNOWN</b> (0x53) Endpoint index does not exist.</p>

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08



*The complete list of the USBX Host Stack Services and their descriptions can be found in chapter 4 of the USB Host Stack User Guide [D1]*

#### 4.2.4. USB Host Classes

The Class APIs are very specific to each USB class. Most of the common APIs for USB classes provide services such as opening/closing a device and reading from and writing to a device.

##### **Storage Class**


The Storage class is in charge of enumerating and driving connected mass storage devices. It is integrated with Eclipse FileX, the file system support module in Eclipse ThreadX, to offer services to operate the connected media through FX\_MEDIA interface.


##### **CDC Class**

The CDC class is in charge of enumerating and driving connected CDC devices. Currently two functionalities are supported:

- CDC-ACM, the serial converter. Following table lists a subset of the USBX Device CDC-ACM Class APIs:


NOT FOR USE

	<b>Safety Manual</b>	Document Status	Released
	<b>USBX Version 6</b>	Document Version	1.0
A high-performance USB stack from Eclipse Foundation		Release Date	2024.07.08

Prototype	Description
<pre> UINT ux_host_class_cdc_acm_read(UX_HOST_CLASS_CDC_ACM *cdc_acm,     UCHAR *data_pointer,     ULONG requested_length,     ULONG *actual_length) </pre>	<p>This function reads from the cdc_acm interface. The call is blocking and only returns when there is either an error or when the transfer is complete.</p> <p><b>Return Values</b></p> <p><b>UX_SUCCESS</b> (0x00) The data transfer was completed.</p> <p><b>UX_Error</b> (0X00) Error from function</p> <p><b>UX_HOST_CLASS_INSTANCE_UNK-NOWN</b> (0x5b) The cdc_acm instance is invalid</p> <p><b>UX_TRANSFER_TIMEOUT</b> (0X5c) Transfer timeout, reading incomplete</p> <p><b>UX_TRANSFER_BUFFER_OVERFLOW</b> (0X27) Transfer buffer overflow, inside a USB packet, host sending more bytes than available buffer</p> <p> <i>Note the function reads bytes from the device packet by packet. If the prepared buffer size is smaller than a packet and the device sends more data than expected (in other words, the prepared buffer size is not a multiple of the USB endpoint's max packet size), then buffer overflow will occur. To avoid this issue, the recommended way to read is to allocate a buffer exactly one packet size (USB endpoint max packet size). This way if there is more data, the next read can get it and no buffer overflow will occur. If there is less data, the current read can get a short packet instead of generating an error</i></p>
<pre> UINT ux_host_class_cdc_acm_ioctl (UX_HOST_CLASS_CDC_ACM *cdc_acm,     ULONG ioctl_function,     VOID *parameter) </pre>	<p>This function is called when an application needs to perform various ioctl calls to the cdc_acm interface.</p> <p><b>Return Values</b></p> <p><b>UX_SUCCESS</b> (0x00) The data transfer was completed</p> <p><b>UX_MEMORY_INSUFFICIENT</b> (0x12) Not enough memory.</p> <p><b>UX_HOST_CLASS_INSTANCE_UNK-NOWN</b> (0x5b) TCDC-ACM instance is in an invalid state</p> <p><b>UX_FUNCTION_NOT_SUPPORTED</b> (0x54) Unknown IOCTL function</p>



Refer to chapter 5 of the “USBX Host User Guide” [D1] for the complete list of the CDCACM class APIs and their descriptions.

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

- CDC-ECM, the ethernet controller. This class is designed to be used with NetX, specifically, the USBX CDC-ECM class acts as the driver for NetX.



Refer to *USB Host CDC-ECM Class* in Chapter 5 of “*USBX Host User Guide*” [D1] for more information

### HID Class

The HID class is in charge of enumerating and driving connect HID devices. The following functionalities are supported by its client services:

- HID Keyboard
- HID Mouse
- HID Remote control



Refer to *HID part* in Chapter 5 of the *USBX Host Device Guide* [D1] for more information.

### HUB Class

The hub class is in charge of driving USB hubs. A USB hub can either be a stand-alone hub or as part of a compound device such as a keyboard or a monitor. A hub can be self-powered or bus powered. Bus-powered hubs have a maximum of four downstream ports and can only allow for the connection of devices that are either self-powered or bus-powered devices that use less than 100mA of power. Hubs can be cascaded. Up to five hubs can be connected to one another.

#### 4.2.5. Host Controller

The USB host controller driver interoperates USB Host Stack operations to hardware actions. Normally, a USBX application should not have to call host controller APIs, except initialization function. When the host controller initialization function is called, the hardware is ready and the stack will reply to detect bus events and start communication with connected devices.


Each host controller instance has one or more USB root hubs. The number of root hubs is either determined by the nature of the controller or can be retrieved by reading specific registers from the controller.

The host controller driver is responsible for driving a specific type of USB controller. A USB host controller can have multiple controllers inside. For instance, certain Intel PC chipset contain two UHCI controllers. Some USB 2.0 controllers contain multiple instances of an OHCI controller in addition to one instance of the EHCI controller.

The Host controller will manage multiple instances of the same controller only. In order to drive most USB 2.0 host controllers, it will be required to initialize both the OCHI controller and the EHCI controller during the initialization of USBX.

The host controller is responsible for managing the following:



	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
A high-performance USB stack from Eclipse Foundation		<i>Release Date</i>	2024.07.08

- **Root Hub:** The root hub management is responsible for the powering up of each controller port and determining if there is a device inserted or not. This functionality is used by the USBX generic root hub to interrogate the controller downstream ports.
- **Power management:** The power management processing provides for the handling of suspend/resume signals either in gang mode, therefore affecting all controller downstream ports at the same time, or individually if the controller offers this functionality.
- **Endpoints:** The endpoint management provides for the creation or destruction of physical endpoints to the controller. The physical endpoints are memory entities that are parsed by the controller if the controller supports master DMA or that are written in the controller. The physical endpoints contain transactions information to be performed by the controller.
- **Transfers:** Transfer management provides for a class to perform a transaction on each of the endpoints that have been created. Each logical endpoint contains a component called TRANSFER REQUEST for USB transfer requests. The TRANSFER REQUEST is used by the stack to describe the transaction. This TRANSFER REQUEST is then passed to the stack and to the controller, which may divide it into several sub transactions depending on the capabilities of the controller.

#### 4.2.6. USB Device Framework

A USB device is represented by a tree of descriptors. There are six main types of descriptors:

- **Device descriptors:** Each USB device has one single device descriptor. This descriptor contains the device identification, the number of configurations supported, and the characteristics of the default control endpoint used for configuring the device.



*Refer to section “Device Descriptors” in chapter 4 of the USBX User Guide [D1] for more details.*

- **Configuration descriptors:** The configuration descriptor describes information about a specific device configuration. A USB device may contain one or more configuration descriptors. The bNumConfigurations field in the device descriptor indicates the number of configuration descriptors. The descriptor contains a bConfigurationValue field with a value that, when used as a parameter to the Set Configuration request, causes the device to assume the described configuration.

The descriptor describes the number of interfaces provided by the configuration. Each interface represents a logical function within the device and may operate independently. For instance, a USB audio speaker may have three interfaces, one for audio streaming, one for audio control, and one HID interface to manage the speaker’s buttons.


When the host issues a GET\_DESCRIPTOR request for the configuration descriptor, all related interface and endpoint descriptors are returned.



*Refer to section “Configuration Descriptors” in chapter 4 of the USBX User Guide [D1] for more details.*

- **Interface descriptors:** The interface descriptor describes a specific interface within a configuration. An interface is a logical function within a USB device. A configuration provides



	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
A high-performance USB stack from Eclipse Foundation		<i>Release Date</i>	2024.07.08

one or more interfaces, each with zero or more endpoint descriptors describing a unique set of endpoints within the configuration. When a configuration supports more than one interface, the endpoint descriptors for a particular interface follow the interface descriptor in the data returned by the GET\_DESCRIPTOR request for the specified configuration.

An interface descriptor is always returned as part of a configuration descriptor. An interface descriptor cannot be directly access by a GET\_DESCRIPTOR request.

An interface may include alternate settings that allow the endpoints and/or their characteristics to be varied after the device has been configured. The default setting for an interface is always alternate setting zero. A class can select to change the current alternate setting to change the interface behaviour and the characteristics of the associated endpoints. The SET\_INTERFACE request is used to select an alternate setting or to return to the default setting. Alternate settings allow a portion of the device configuration to be varied while other interfaces remain in operation. If a configuration has alternate settings for one or more of its interfaces, a separate interface descriptor and its associated endpoints are included for each setting.



*Refer to section “Interface Descriptors” in chapter 4 of the USBX Guide [D1] for more details.*

- Endpoint descriptors: Each endpoint associated with an interface has its own endpoint descriptor. This descriptor contains the information required by the host stack to determine the bandwidth requirements of each endpoint, the maximum payload associated with the endpoint, its periodicity, and its direction. An endpoint descriptor is always returned by a GET\_DESCRIPTOR command for the configuration.

The default control endpoint associated with the device descriptor is not counted as part of the endpoint(s) associated with the interface and therefore not returned in this descriptor. When the host software requests a change of the alternate setting for an interface, all the associated endpoints and their USB resources are modified according to the new alternate setting.



*Except for the default control endpoints, endpoints cannot be shared between interfaces.*

*Refer to section “Endpoint Descriptors” in chapter 4 of the USBX Host Device Guide [D1] for more details.*

- String descriptors: String descriptors are optional. If a device does not support string descriptors, all references to string descriptors within device, configuration, and interface descriptors must be reset to zero. String descriptors use UNICODE encoding, thus allowing the support for several character sets. The strings in a USB device may support multiple languages. When requesting a string descriptor, the requester specifies the desired language using a language ID defined by the USB-IF.



*To get the latest LANGID definitions go to*


*<https://docs.microsoft.com/enus/windows/desktop/intl/language-identifier-constants-and-strings>.*

*This page will change as new LANGIDs are added.*



*Note that the HID Primary LANGID (0x0FF) is not on the above list, however it is permanently reserved and will never be reassigned.*

String index zero for all languages returns a string descriptor that contains an array of two-byte LANGID codes supported by the device. It should be noted that the UNICODE string is not 0

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
A high-performance USB stack from Eclipse Foundation		<i>Release Date</i>	2024.07.08

terminated. Instead, the size of the string array is computed by subtracting two from the size of the array contained in the first byte of the descriptor.



Refer to section “String Descriptors” in chapter 4 of the *USBX Host Device Guide [D1]* for more details.

- **Functional descriptors:** Functional descriptors are also known as class-specific descriptors. They normally use the same basic structures as generic descriptors and allow for additional information to be available to the class. For example, in the case of the USB audio speaker, class specific descriptors allow the audio class to retrieve for each alternate setting the type of audio frequency supported.

USBX maintains most device descriptors in memory, that is, all descriptors except the string and functional descriptors.



Section “USBX Device Descriptor Framework in Memory” of chapter 4 of the *USBX Host Device Guide [D1]* includes a diagram showing how these descriptors are stored and related.

## 5. QUALIFICATION OF THE USBX SOFTWARE

### 5.1. Quality Management System

The Eclipse Foundation’s quality policy is explained in “USBX Quality Management Document” [D5].


Eclipse Foundation has established, documented, implemented, and currently maintains a quality management system. The quality management effectiveness is continually improved in accordance with the requirements defined by this organization.

Eclipse Foundation:

- has identified the processes needed for the quality management system and their application throughout the organization,
- determined the sequence and interaction of these processes,
- determined criteria and methods needed to ensure that both the operation and control of these processes are effective,
- ensures the availability of resources and information necessary to support the operation and monitoring of these processes,
- monitors, measures and analyses these processes, and
- implements actions necessary to achieve planned results and continual improvement of these processes.

These processes are managed by the organization in accordance with the requirements defined by this organization.

Where Eclipse Foundation chooses to outsource any process that affects product conformity with requirements, the organization ensures control over such processes. Controls of such outsourced processes are identified within the quality management system.


	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

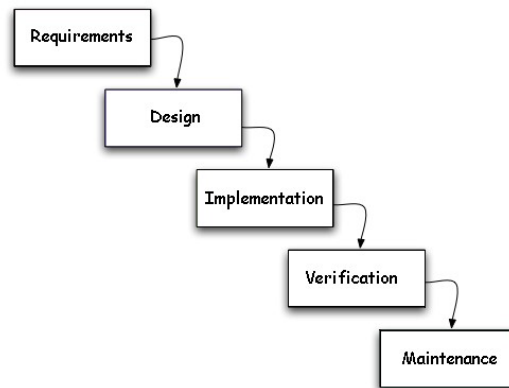
“USBX Quality Management Document” [D5] addresses following issues:

- Quality Management System
  - General requirements
  - Documentation requirements
- Management Responsibility
  - Management commitment
  - Customer focus
  - Quality policy
  - Planning
  - Responsibility, authority and communication
  - Management review
- Resource Management
  - Provision of resources
  - Competence, awareness, and training - Infrastructure and work environment
- Product Realization
  - Planning of product realization
  - Customer-related processes
  - Design and development process
  - Purchasing process
  - Production and service provision
  - Control of monitoring and measuring devices
- Measurement, Analysis, and Improvement
  - Monitoring and measurement of the quality management performance
  - Control of nonconforming products
  - Data analysis
  - Improvement

## 5.2. USBX Quality Assurance

Eclipse USBX utilizes a modified waterfall model for product development with phase overlap and blending.

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08



**Figure 4: Shell window (excerpt) of the USBX 6.1.11 test suite**

The USBX Quality Document [D3] describes company guidelines on the product development phases of the USBX software:

#### a) Requirements

The requirements for each new USBX version require analysis of the defects as well as features asked by customers and marketing. Defects that are verified are always fixed in the new version. Feature enhancements are subject to a cost-benefit analysis. Note that this analysis may include fast prototyping. Naturally, not all new features are incorporated into each new release.

#### b) Design


USBX has three principal design goals: simplicity, scalability in size and high performance. In many situations these goals are complementary, i.e. simpler, smaller software usually gives better performance.

#### c) Implementation

USBX is implemented as a C library, which must be linked with the application software. The USBX library typically consists of 313 object files that are derived from 313 C source files. There are also 17 C include files that are used in the C file compilation process. All the C source and include files conform completely to the ANSI C standard.

USBX applications need access to two include files: `ux_api.h` and `ux_port.h`. The `ux_api.h` file contains all the constants, function prototypes, and object data structures. This file is generic; i.e., it is typically the same for all processor support packages. The `ux_port.h` file is included by `ux_api.h`. It contains processor and/or development tool specific information, including data type assignments and interrupt management macros that are used throughout the USBX C source code. The `ux_port.h` file also contains the USBX Duo port-specific ASCII version string `_ux_version_id`.

Eclipse USBX utilizes a software component methodology in its products. A software component is somewhat similar to an object or class in C++. Each component provides a set of action functions that operate on the internal data of the component. In general, components are not allowed access to the global data of other components. The only exception to this rule is the System component. If it were not for the design goal of scalable code size, component files would likely contain more than

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
	A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.07.08

one function. In general, Eclipse USBX recommends a “more than one function per-file” approach to application development.

All USBX software conforms to a strict set of coding conventions. This makes it easier to understand and maintain. In addition, it provides a reasonable template for application software conventions. The coding conventions concern:

- USBX file names
- USBX name space
- USBX constant names
- USBX Struct and Typedef Names
- USBX Member Names
- USBX Local Data Names
- USBX Global Data Names
- USBX Function Names
- Source Code Indentation
- Comments

#### d) Verification

Each new version of USBX is subject to extensive verification prior to becoming an official release, as detailed in “USBX Software Test Document” [D2]. This document describes the test suite and documents the results of running the test along with the code coverage analysis. Each USBX version must achieve 100% code coverage.

For verification details refer to chapter 5.3 of this document.

#### e) Maintenance

Defects and new feature requests are maintained in DevOps per Eclipse ThreadX development process. All defects and new feature requests are maintained in this database.

Eclipse USBX source code version control is also accomplished via the standard source code management system DevOps. Read privileges are given to all developers. However, write privilege for each release is given only to the Eclipse ThreadX Project Lead.

### 5.3. Verification of the USBX Software


#### 5.3.1. Master Test Plan for the USBX 6.1.11 USB access system

The USBX 6.1.11 test suite is comprised of 386 manual test cases that effectively perform functional “black box” test over the entire USBX 6.1.11 library routines.

Each of the 386 tests in the USBX 6.1.11 test suite outputs the type of test performed as well as the result of the test. Valid results are “SUCCESS” for successful completion, “ERROR” for unsuccessful completion.

All services and features of the USBX 6.1.11 are tested and all test cases (i.e. 386) of the USBX test suite were passed.



	<b>Safety Manual</b>	Document Status	Released
	<b>USBX Version 6</b>	Document Version	1.0
A high-performance USB stack from Eclipse Foundation		Release Date	2024.07.08

```

1/386 Test #4: default_build_coverage::usbx_ux_api_tracex_id_test ..... Passed 0.03 sec
2/386 Test #3: default_build_coverage::usbx_ux_host_stack_device_string_get_test ..... Passed 0.03 sec
3/386 Test #5: default_build_coverage::usbx_ux_utility_descriptor_pack_test ..... Passed 0.02 sec
4/386 Test #6: default_build_coverage::usbx_ux_utility_descriptor_parse_test ..... Passed 0.02 sec
5/386 Test #7: default_build_coverage::usbx_ux_utility_memory_safe_test ..... Passed 0.03 sec
6/386 Test #9: default_build_coverage::usbx_ux_utility_pci_write_test ..... Passed 0.02 sec
7/386 Test #10: default_build_coverage::usbx_ux_utility_pci_read_test ..... Passed 0.02 sec
8/386 Test #8: default_build_coverage::usbx_ux_utility_memory_test ..... Passed 0.09 sec
9/386 Test #11: default_build_coverage::usbx_ux_utility_pci_class_scan_test ..... Passed 0.03 sec
10/386 Test #12: default_build_coverage::usbx_ux_utility_physical_address_test ..... Passed 0.03 sec
11/386 Test #13: default_build_coverage::usbx_ux_utility_string_length_check_test ..... Passed 0.03 sec
12/386 Test #14: default_build_coverage::usbx_ux_utility_unicode_to_string_test ..... Passed 0.02 sec
13/386 Test #15: default_build_coverage::usbx_ux_utility_event_flags_test ..... Passed 0.03 sec
14/386 Test #16: default_build_coverage::usbx_ux_utility_mutex_test ..... Passed 0.03 sec
374/386 Test #376: default_build_coverage::usbx_ux_host_class_storage_media_open_test ..... Passed 0.08 sec
375/386 Test #377: default_build_coverage::usbx_ux_host_class_storage_media_read_test ..... Passed 0.06 sec
376/386 Test #378: default_build_coverage::usbx_ux_host_class_storage_media_write_test ..... Passed 0.10 sec
377/386 Test #361: default_build_coverage::usbx_ux_device_class_storage_thread_test ..... Passed 0.70 sec
378/386 Test #379: default_build_coverage::usbx_ux_host_class_storage_media_protection_check_test ..... Passed 0.29 sec
379/386 Test #380: default_build_coverage::usbx_ux_host_class_storage_media_recovery_sense_get_test ..... Passed 0.25 sec
380/386 Test #381: default_build_coverage::usbx_ux_host_class_storage_start_stop_test ..... Passed 0.31 sec
381/386 Test #383: default_build_coverage::usbx_ux_host_class_storage_driver_entry_test ..... Passed 0.12 sec
382/386 Test #366: default_build_coverage::usbx_ux_device_class_storage_invalid_lun_test ..... Passed 0.94 sec
383/386 Test #384: default_build_coverage::usbx_ux_host_class_storage_entry_test ..... Passed 0.07 sec
384/386 Test #382: default_build_coverage::usbx_ux_host_class_storage_transport_bo_test ..... Passed 0.42 sec
385/386 Test #386: default_build_coverage::usbx_ux_host_class_storage_fats_exfat_test ..... Passed 0.76 sec
386/386 Test #385: default_build_coverage::usbx_ux_host_class_storage_thread_entry_test ..... Passed 6.34 sec
100% tests passed, 0 tests failed out of 386
Total Test time (real) = 20.56 sec

```

**Figure 4: Shell window (excerpt) of the USBX 6.1.11 test suite**



More details to test environment, planned and performed verification activities and verification deliverables can be found the Master test plan for the USBX 6.1.11 USB access system (i.e. USBX IEEE829 Software Test Document [D3]).

### 5.3.2. Test Code Coverage Analysis

The test suite is executed on a PC host running Ubuntu 18.04.3 LTS operating system. The compiler being used to generate validation executable is gcc: gcc (Ubuntu 7.5.0-3ubuntu1~18.04)7.5.0)

gcov provides a coverage analysis. The version is: gcov (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0

USBX generic code coverage is performed by gcov test coverage suite.


The test was performed on 7424 lines and 3295 branches and shows 100% line / branch coverage for each file (refer to USBX IEEE829 Software Coverage Report [D6] for more details).

### 5.3.3. USBX Static Analysis

IAR C-STAT test workbench is used for the static analysis. 807 source files, 784 functions and 51 Header files were analysed. No issues were found (refer to USBX CSTAT Report [D7] for the details of the test results

## 5.4. Certification of USBX 6.1.11

For the USBX certification performed by third party bodies refer to chapter 1.4 of this safety manual.

	<b>Safety Manual</b>	<i>Document Status</i>	Released
	<b>USBX Version 6</b>	<i>Document Version</i>	1.0
A high-performance USB stack from Eclipse Foundation		<i>Release Date</i>	2024.07.08

## 6. CONCLUSION

### 6.1. Backward Compatibility

USBX is backward compatible. Applications written for earlier versions of USBX will run without change on the latest version.

The modifications done on new USBX versions are described in the code version information. Users have to analyse the modifications to understand the differences between releases, since the backward compatibility is guaranteed.

### 6.2. Compatibility with other systems

There are no USBX versions build on special pre-conditions.

### 6.3. Requirements not met

Requirements for each new version of USBX require an analysis of the defects as well as the “wish list” features asked for by customers and marketing. Defects that are verified are always fixed in the new version. Feature enhancements are subject to a cost-benefit analysis.

Note that this analysis may include fast prototyping. Naturally, not all new features are incorporated into each new release.

All changes (e.g. enhancements, clean ups, bug fixes, new features), which have been implemented since revision 6.0 are documented in the USBX Maintenance Report [D4].

### 6.4. Outstanding Anomalies

Each new USBX version is subject to extensive verification prior to becoming an official release.

Possible anomalies are either found internally by running the test suite comprised of 386 test cases that effectively perform functional “black box” test over the entire USBX 6.1.11 library routines or via reported customer problem.

Founded anomalies are recorded and their corrections are incorporated in the next new USBX version.

According to the USBX IEEE829 Software Test document [D2] and USBX Maintenance Report [D4] there are no outstanding anomalies in the current USBX version (i.e. version 6, revision 6.1.11).


### 6.5. Design Safe State

IEC 61508 defines a Safe State as “state of the Equipment Under Control when safety is achieved”. IEC 61508-3 Annex D: “In certain circumstances, upon controlled failure of the system application, the element (as an element a Hardware or Software element is meant) may revert to a design safe state. In such circumstances, the precise definition of design safe state should be specified for consideration by the integrator”.

ISO 26262 defines a Safe State as “operating mode of an item without an unreasonable level of risk”.

The term Safe State does not concern USBX but furthermore it concerns the safety application using the underlying USBX services.



	<b>Safety Manual</b>	Document Status	Released
		Document Version	1.0
	<b>USBX Version 6</b>	Release Date	2024.05.17
	A high-performance USB stack from Eclipse Foundation		

## 6.6. Functional Safety View

The safety standards require for software architecture design beside completeness and correctness with respect to software requirement specification, freedom of intrinsic design faults, testability, simplicity, understandability and predictable behaviour a fault tolerance (control of failures) in addition.



*Some examples of software architecture design requirements are given in:*  
*Table A.2 of IEC 61508-3 [S1]*  
*Table 4 of ISO 26262-6 [S3] and*  
*Table 3 of EN 50128 [S4]*

***Fault detection and diagnosis provides the basis for functional safety countermeasures to minimize the consequences of failure.***

## 6.7. Data Integrity

Data integrity is fundamental component of information security and thereby safety.

In its broadest use, “data integrity” refers to the accuracy and consistency of data:

- received from the sensor(s)
- stored in volatile and non-volatile memory
- processed by CPU
- received via external communication interfaces
- exchanged on internal system interfaces
- data entered by the user

In view of the USBX application the data integrity concerns safety relevant data entered by the user, safety data exchanged on internal system interfaces, safety data processed and stored in volatile and non-volatile memory.

*In view of higher Safety Integrity Levels (SIL) additional safety mechanisms to achieve high data integrity are to be considered in the safety related application using USBX services.*


*Examples on some measures are given below:*



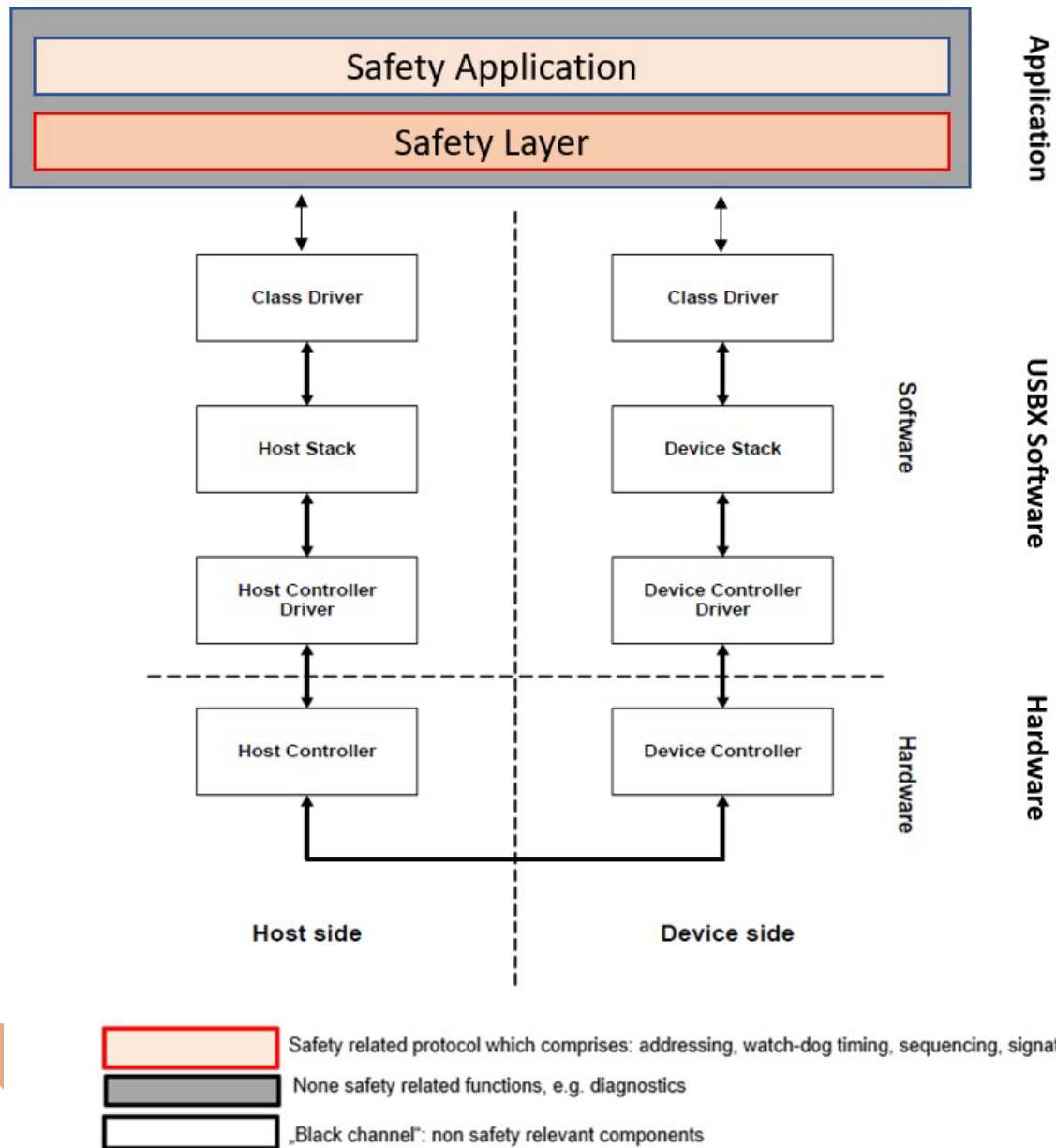
- *Securing of the USBX services by program flow monitoring*
- *Protection of safety related data by memory signature / data replication*
- *Validation of safety data entered by the user (range and plausibility checks, re-prompting for invalid inputs, user prompting for data confirmation)*
- *Verification of data content integrity by information redundancy, data timing and sequencing*

## 6.8. Interface between the safety related application and USBX stack

If USBX will be used in a safety related context, it can be beneficial to implement a safety layer between the safety related application and the USBX Software. That safety layer acts as an error screen giving useful information to the user when a failure is detected and would be analogous to dialog screen applying when Window application fails.


	<b>Safety Manual</b>	<i>Document Status</i>	Released
		<i>Document Version</i>	1.0
	<b>USBX Version 6</b> A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.05.17

The figure below shows the safety layer embedded between the safety related application and USBX stack.



**Figure 5: Usage of the USBX Software in a safety related context**

The safety application itself has to implement failsafe principle i.e. entering of safe state for cases when the timely delivery of messages is disturbed, delayed or not possible at all.

	<b>Safety Manual</b>	<i>Document Status</i>	Released
		<i>Document Version</i>	1.0
	<b>USBX Version 6</b> A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.05.17

The safety layer (or alternatively the safety application) has to evaluate USBX fault handling by:

- *Checking of the return code from the USBX API*
- *Cyclic monitoring the USBX system error information.*


*Note: USBX checks for internal system errors by setting breakpoint at the function calls `_ux_system_error_handler`.*



*The safety layer (or related application) should monitor the internal system status to determine if any internal errors in USBX have taken place*

*Creating of fault information screen for visual reporting of USBX interface and general application fault information. When an applicable fault is detected, the user application should use USBX to present this fault screen in addition to the internal fault processing in the application*

NOT FOR USE

	<b>Safety Manual</b>	<i>Document Status</i>	Released
		<i>Document Version</i>	1.0
	<b>USBX Version 6</b> A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.05.17

## APPENDICES


### Appendix [A]: TERMS AND ACRONYMS

#### [A1] Reference Documents

Document ID / (file)	Revision / Date
[D1] USBX: User Guide (Version 6) ( <a href="https://github.com/eclipse-threadx/rtos-docs/blob/main/rtos-docs/usbx/index.md">https://github.com/eclipse-threadx/rtos-docs/blob/main/rtos-docs/usbx/index.md</a> )	April 2022
[D2] USBX IEEE829 Software Test Document (USBX_IEEE829_Software_Test_Document_04_25_2022.pdf)	April 25, 2022
[D3] USBX Quality Document (USBX_Quality_Document_04_25_2022)	April 25, 2022
[D4] USBX Maintenance Report (USBX_Maintenance_Report_04_25_2022.pdf)	April 25, 2022
[D5] USBX Quality Management Document (USBX_Quality_Management_Document_04_25_2022.pdf)	April 25, 2022
[D6] USBX IEEE829 Software Coverage Report (USBX_IEEE829_Software_Coverage_Report_04_25_2022.pdf)	April 19, 2022
[D7] USBX CSTAT Report (USBX_CStat_report_08_15_2019.pdf)	Aug 15, 2019
[D8] Eclipse ThreadX: User Guide (Version 6) ( <a href="https://github.com/eclipse-threadx/rtos-docs/blob/main/rtos-docs/threadx/index.md">https://github.com/eclipse-threadx/rtos-docs/blob/main/rtos-docs/threadx/index.md</a> )	October 2020
[D9] Certification Report for Functional Safety Report No.: P2GF0004	July 2022

#### [A2] Acronyms

Acronym	Extended meaning
API	Application Program Interface
ASIL	Automotive Safety Integrity Level

	<b>Safety Manual</b>	<i>Document Status</i>	Released
		<i>Document Version</i>	1.0
	<b>USBX Version 6</b> A high-performance USB stack from Eclipse Foundation	<i>Release Date</i>	2024.05.17

BDM	Background Debug Mode
GUI	Graphical User Interface
ICE	In-circuit Emulation
I/O	Input / Output
Mutex	Mutual Exclusion
RAM	Random Access Memory
ROM	Read Only Memory
RTOS	Real-Time Operating System
SIL	Safety Integrity Level

NOT FOR USE