

**SysToMath IO C++ Libraries Implementation Reference Manual**  
Version 1.07-r348

Generated by Doxygen 1.5.4-20071203

Thu Jan 3 20:40:51 2008

## Contents

<b>1</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Main Page</a></b>	<b>1</b>
<b>2</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Module Index</a></b>	<b>1</b>
<b>3</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Directory Hierarchy</a></b>	<b>2</b>
<b>4</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Namespace Index</a></b>	<b>2</b>
<b>5</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Class Index</a></b>	<b>2</b>
<b>6</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Class Index</a></b>	<b>3</b>
<b>7</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation File Index</a></b>	<b>4</b>
<b>8</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Module Documentation</a></b>	<b>5</b>
<b>9</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Directory Documentation</a></b>	<b>10</b>
<b>10</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Namespace Documentation</a></b>	<b>15</b>
<b>11</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Class Documentation</a></b>	<b>16</b>
<b>12</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation File Documentation</a></b>	<b>86</b>
<b>13</b>	<b><a href="#">SysToMath IO C++ Libraries Implementation Page Documentation</a></b>	<b>102</b>

## **1 SysToMath IO C++ Libraries Implementation Main Page**

### **1.1 Introduction**

This documentation describes the C++ libraries contained in the SysToMath IO C++ Libraries package:

- [SysToMath Device C++ Library \(Headers only\)](#)
- [SysToMath W32Device C++ Library](#)
- [SysToMath UsbDevice C++ Library](#)

### **1.2 Supported Tool Families**

The C++ libraries contained in the SysToMath IO C++ Libraries package are designed to support the tool families:

- [Microsoft Visual Studio Tool Family](#)
- [GNU Tool Family](#)

## 2 SysToMath IO C++ Libraries Implementation Module Index

### 2.1 SysToMath IO C++ Libraries Implementation Modules

Here is a list of all modules:

<b>SysToMath Device C++ Library</b>	<b>5</b>
<b>Device: Abstract Base Class for Generic Devices</b>	<b>5</b>
<b>Device Implementation</b>	<b>6</b>
<b>SysToMath USB Device C++ Library</b>	<b>7</b>
<b>UsbDevice: Base Class for USB Devices</b>	<b>8</b>
<b>UsbDevice Implementation</b>	<b>7</b>
<b>SysToMath Win32 Device C++ Library</b>	<b>9</b>
<b>W32Device: Base Class for Win32 Devices</b>	<b>10</b>
<b>W32Device Implementation</b>	<b>9</b>

## 3 SysToMath IO C++ Libraries Implementation Directory Hierarchy

### 3.1 SysToMath IO C++ Libraries Implementation Directories

This directory hierarchy is sorted roughly, but not completely, alphabetically:

<b>stmdevice</b>	<b>14</b>
<b>stm</b>	<b>13</b>
<b>impl</b>	<b>10</b>
<b>internal</b>	<b>12</b>
<b>stmusbdevice</b>	<b>14</b>
<b>stm</b>	<b>13</b>
<b>impl</b>	<b>11</b>
<b>stmw32device</b>	<b>14</b>
<b>stm</b>	<b>12</b>
<b>impl</b>	<b>11</b>

## 4 SysToMath IO C++ Libraries Implementation Namespace Index

### 4.1 SysToMath IO C++ Libraries Implementation Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">stm</a>	15
<a href="#">stm::local</a>	15

## 5 SysToMath IO C++ Libraries Implementation Class Index

### 5.1 SysToMath IO C++ Libraries Implementation Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<a href="#">stm::Device::Descriptor</a>	35
<a href="#">stm::Device::Version</a>	35
<a href="#">stm::UsbCtrl</a>	39
<a href="#">stm::UsbDevice::InterfaceClass</a>	56
<a href="#">stm::local::UsbHandle</a>	57
<a href="#">stm::local::UsbHandle::Lock</a>	58
<a href="#">stm::UsbPipe</a>	59
<a href="#">stm::Uuid</a>	61
<a href="#">stm::W32Device::InterfaceClass</a>	76
<a href="#">stm::local::xDevice</a>	77
<a href="#">stm::Device</a>	16
<a href="#">stm::UsbDevice</a>	40
<a href="#">stm::W32Device</a>	65
<a href="#">stm::local::xUsbDevice</a>	80
<a href="#">stm::UsbDevice</a>	40
<a href="#">stm::local::xW32Device</a>	84
<a href="#">stm::W32Device</a>	65
<a href="#">stm::local::xW32Device::Ident</a>	85

## 6 SysToMath IO C++ Libraries Implementation Class Index

### 6.1 SysToMath IO C++ Libraries Implementation Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">stm::Device</a> (Abstract base class defining the C++ API for a generic device )	16
<a href="#">stm::Device::Descriptor</a> (Objects of type <a href="#">Device::Descriptor</a> describe system dependent aspects of a <a href="#">Device</a> as a pair of a void pointer and a void function pointer )	35
<a href="#">stm::Device::Version</a> ( <a href="#">Device</a> driver version )	35
<a href="#">stm::UsbCtrl</a> (Type specifying the request type encoding the transfer direction, the value and the index of a USB control request )	39
<a href="#">stm::UsbDevice</a> (Class defining the C++ API for a libusb controlled USB device inheriting the generic <a href="#">stm::Device</a> C++ API )	40
<a href="#">stm::UsbDevice::InterfaceClass</a> (Type describing a libusb controlled USB device interface class )	56
<a href="#">stm::local::UsbHandle</a>	57
<a href="#">stm::local::UsbHandle::Lock</a>	58
<a href="#">stm::UsbPipe</a> (Type specifying the configuration, the interface, the alternate setting, and the endpoint of the USB pipe to be used for a USB bulk or interrupt transfer )	59
<a href="#">stm::Uuid</a> (Universal unique identifier storing its fields in little endian format )	61
<a href="#">stm::W32Device</a> (Class defining the C++ API for a Windows device inheriting the generic <a href="#">stm::Device</a> C++ API )	65
<a href="#">stm::W32Device::InterfaceClass</a> (Type describing a Windows device interface class )	76
<a href="#">stm::local::xDevice</a>	77
<a href="#">stm::local::xUsbDevice</a>	80
<a href="#">stm::local::xW32Device</a>	84
<a href="#">stm::local::xW32Device::Ident</a>	85

## 7 SysToMath IO C++ Libraries Implementation File Index

### 7.1 SysToMath IO C++ Libraries Implementation File List

Here is a list of all files with brief descriptions:

<a href="#">config.h</a> (Definition of macros for library generation and of pragmas for automatic library choice )	86
<a href="#">device.hpp</a> (Abstract base class <a href="#">stm::Device</a> forming the ANSI-C++ API for generic devices )	87

<a href="#">usbdevice.cpp</a> (Base class <a href="#">stm::UsbDevice</a> forming the ANSI-C++ API for libusb controlled USB devices )	89
<a href="#">usbdevice.hpp</a> (Base class <a href="#">stm::UsbDevice</a> forming the ANSI-C++ API for libusb controlled USB devices )	90
<a href="#">usbdeviceconfig.h</a> (Management of library generation and of automatic library choice for the Artista W32 SDK library stmusbdevice )	92
<a href="#">w32device.cpp</a> (Base class <a href="#">stm::W32Device</a> forming the ANSI-C++ API for Win32 devices )	93
<a href="#">w32device.hpp</a> (Base class <a href="#">stm::W32Device</a> forming the ANSI-C++ API for Win32 devices )	94
<a href="#">w32deviceconfig.h</a> (Management of library generation and of automatic library choice for the Artista W32 SDK library stmw32device )	95
<a href="#">xdevice.hpp</a> (Abstract base class <a href="#">stm::Device</a> forming the ANSI-C++ API for generic devices )	97
<a href="#">xusbdevice.hpp</a> (Base class <a href="#">stm::UsbDevice</a> forming the ANSI-C++ API for libusb controlled USB devices )	99
<a href="#">xw32device.hpp</a> (Base class <a href="#">stm::W32Device</a> forming the ANSI-C++ API for Win32 devices )	100

## 8 SysToMath IO C++ Libraries Implementation Module Documentation

### 8.1 SysToMath Device C++ Library

Collaboration diagram for SysToMath Device C++ Library:



#### 8.1.1 Detailed Description

SysToMath Device C++ Library (stmdevice).

The SysToMath Device C++ Library consists of C++ header files providing the abstract base class [stm::Device](#) forming the ANSI-C++ API for generic devices.

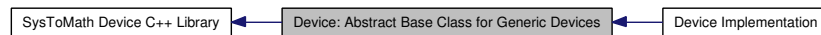
#### Modules

- [Device: Abstract Base Class for Generic Devices](#)

*Abstract base class for generic devices neutralizing the native operating system interfaces.*

## 8.2 Device: Abstract Base Class for Generic Devices

Collaboration diagram for Device: Abstract Base Class for Generic Devices:



### 8.2.1 Detailed Description

Abstract base class for generic devices neutralizing the native operating system interfaces.

#### Files

- file [device.hpp](#)  
*Abstract base class `stm::Device` forming the ANSI-C++ API for generic devices.*

#### Modules

- [Device Implementation](#)  
*Implementation of the abstract base class for generic devices neutralizing the native operating system interfaces.*

#### Classes

- struct [stm::Uuid](#)  
*Universal unique identifier storing its fields in little endian format.*
- class [stm::Device](#)  
*Abstract base class defining the C++ API for a generic device.*
- struct [stm::Device::Version](#)  
*Device driver version.*

#### Functions

- `std::ostream & stm::operator<< (std::ostream &os, const Device &device)`  
*Insert a description of the Device device into os.*

### 8.2.2 Function Documentation

#### 8.2.2.1 `std::ostream & stm::operator<< (std::ostream & os, const Device & device)` [inline]

Insert a description of the Device *device* into *os*.

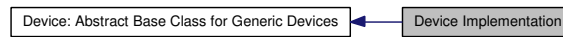
The function inserts a verbal description of the [Device](#) *device* into the output stream *os* and returns *os*.

Definition at line 1179 of file xdevice.hpp.

References `stm::Device::describe()`.

## 8.3 Device Implementation

Collaboration diagram for Device Implementation:



### 8.3.1 Detailed Description

Implementation of the abstract base class for generic devices neutralizing the native operating system interfaces.

#### Files

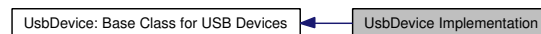
- file [xdevice.hpp](#)  
*Abstract base class `stm::Device` forming the ANSI-C++ API for generic devices.*

#### Classes

- class `stm::local::xDevice`

## 8.4 UsbDevice Implementation

Collaboration diagram for UsbDevice Implementation:



### 8.4.1 Detailed Description

Implementation of the base class for USB devices neutralizing the native operating system interfaces.

#### Files

- file [usbdeviceconfig.h](#)  
*Management of library generation and of automatic library choice for the Artista W32 SDK library `stmusbdevice`.*
- file [xusbdevice.hpp](#)  
*Base class `stm::UsbDevice` forming the ANSI-C++ API for `libusb` controlled USB devices.*
- file [usbdevice.cpp](#)  
*Base class `stm::UsbDevice` forming the ANSI-C++ API for `libusb` controlled USB devices.*

**Classes**

- class [stm::local::xUsbDevice](#)

**8.5 SysToMath USB Device C++ Library**

Collaboration diagram for SysToMath USB Device C++ Library:

**8.5.1 Detailed Description**

SysToMath USB Device C++ Library (stmusbdevice).

The SysToMath USB Device C++ Library consists of a library object providing the base class [stm::UsbDevice](#) forming the ANSI-C++ API for USB devices.

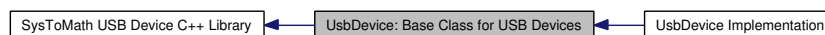
**Modules**

- [UsbDevice: Base Class for USB Devices](#)

*Base class for USB devices neutralizing the native operating system interfaces.*

**8.6 UsbDevice: Base Class for USB Devices**

Collaboration diagram for UsbDevice: Base Class for USB Devices:

**8.6.1 Detailed Description**

Base class for USB devices neutralizing the native operating system interfaces.

**Files**

- file [usbdevice.hpp](#)

*Base class [stm::UsbDevice](#) forming the ANSI-C++ API for libusb controlled USB devices.*

**Modules**

- [UsbDevice Implementation](#)

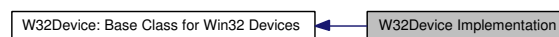
*Implementation of the base class for USB devices neutralizing the native operating system interfaces.*

## Classes

- struct [stm::UsbPipe](#)  
*Type specifying the configuration, the interface, the alternate setting, and the endpoint of the USB pipe to be used for a USB bulk or interrupt transfer.*
- struct [stm::UsbCtrl](#)  
*Type specifying the request type encoding the transfer direction, the value and the index of a USB control request.*
- class [stm::UsbDevice](#)  
*Class defining the C++ API for a libusb controlled USB device inheriting the generic [stm::Device](#) C++ API.*
- struct [stm::UsbDevice::InterfaceClass](#)  
*Type describing a libusb controlled USB device interface class.*

## 8.7 W32Device Implementation

Collaboration diagram for W32Device Implementation:



### 8.7.1 Detailed Description

Implementation of the base class for Win32 devices neutralizing the native operating system interfaces.

## Files

- file [w32deviceconfig.h](#)  
*Management of library generation and of automatic library choice for the Artista W32 SDK library `stmw32device`.*
- file [xw32device.hpp](#)  
*Base class [stm::W32Device](#) forming the ANSI-C++ API for Win32 devices.*
- file [w32device.cpp](#)  
*Base class [stm::W32Device](#) forming the ANSI-C++ API for Win32 devices.*

## Classes

- class [stm::local::xW32Device](#)

## 8.8 SysToMath Win32 Device C++ Library

Collaboration diagram for SysToMath Win32 Device C++ Library:



### 8.8.1 Detailed Description

SysToMath Win32 Device C++ Library (stmw32device).

The SysToMath Win32 Device C++ Library consists of a library object providing the base class [stm::W32Device](#) forming the ANSI-C++ API for Win32 devices.

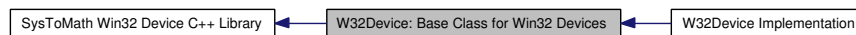
#### Modules

- [W32Device: Base Class for Win32 Devices](#)

*Base class for Win32 devices neutralizing the native operating system interfaces.*

## 8.9 W32Device: Base Class for Win32 Devices

Collaboration diagram for W32Device: Base Class for Win32 Devices:



### 8.9.1 Detailed Description

Base class for Win32 devices neutralizing the native operating system interfaces.

#### Files

- file [w32device.hpp](#)

*Base class [stm::W32Device](#) forming the ANSI-C++ API for Win32 devices.*

#### Modules

- [W32Device Implementation](#)

*Implementation of the base class for Win32 devices neutralizing the native operating system interfaces.*

#### Classes

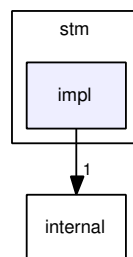
- class [stm::W32Device](#)

*Class defining the C++ API for a Windows device inheriting the generic [stm::Device](#) C++ API.*

- struct [stm::W32Device::InterfaceClass](#)  
*Type describing a Windows device interface class.*

## 9 SysToMath IO C++ Libraries Implementation Directory Documentation

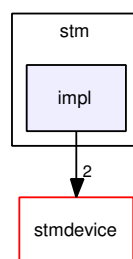
### 9.1 stmdevice/stm/impl/ Directory Reference



#### Files

- file [xdevice.hpp](#)  
*Abstract base class [stm::Device](#) forming the ANSI-C++ API for generic devices.*

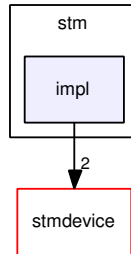
### 9.2 stmw32device/stm/impl/ Directory Reference



#### Files

- file [w32deviceconfig.h](#)  
*Management of library generation and of automatic library choice for the Artista W32 SDK library [stmw32device](#).*
- file [xw32device.hpp](#)  
*Base class [stm::W32Device](#) forming the ANSI-C++ API for Win32 devices.*

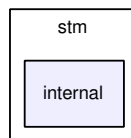
### 9.3 stmusbdevice/stm/impl/ Directory Reference



#### Files

- file [usbdeviceconfig.h](#)  
*Management of library generation and of automatic library choice for the Artista W32 SDK library `stmusbdevice`.*
- file [xusbdevice.hpp](#)  
*Base class `stm::UsbDevice` forming the ANSI-C++ API for libusb controlled USB devices.*

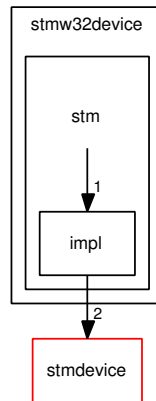
### 9.4 stmdevice/stm/internal/ Directory Reference



#### Files

- file [config.h](#)  
*Definition of macros for library generation and of pragmas for automatic library choice.*

## 9.5 stmw32device/stm/ Directory Reference



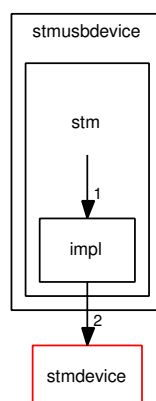
### Directories

- directory [impl](#)

### Files

- file [w32device.cpp](#)  
*Base class `stm::W32Device` forming the ANSI-C++ API for Win32 devices.*
- file [w32device.hpp](#)  
*Base class `stm::W32Device` forming the ANSI-C++ API for Win32 devices.*

## 9.6 stmusbdevice/stm/ Directory Reference



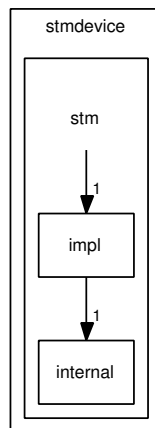
### Directories

- directory [impl](#)

### Files

- file [usbdevice.cpp](#)  
*Base class `stm::UsbDevice` forming the ANSI-C++ API for libusb controlled USB devices.*
- file [usbdevice.hpp](#)  
*Base class `stm::UsbDevice` forming the ANSI-C++ API for libusb controlled USB devices.*

## 9.7 stmdevice/stm/ Directory Reference



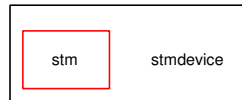
### Directories

- directory [impl](#)
- directory [internal](#)

### Files

- file [device.hpp](#)  
*Abstract base class `stm::Device` forming the ANSI-C++ API for generic devices.*

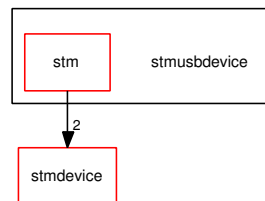
## 9.8 stmdevice/ Directory Reference



### Directories

- directory [stm](#)

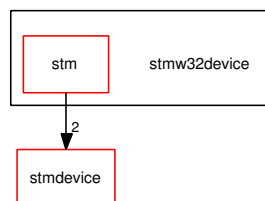
## 9.9 stmusbdevice/ Directory Reference



### Directories

- directory [stm](#)

## 9.10 stmw32device/ Directory Reference



### Directories

- directory [stm](#)

## 10 SysToMath IO C++ Libraries Implementation Namespace Documentation

### 10.1 stm Namespace Reference

#### Classes

- struct [Uuid](#)  
*Universal unique identifier storing its fields in little endian format.*
- class [Device](#)  
*Abstract base class defining the C++ API for a generic device.*
- struct [UsbPipe](#)  
*Type specifying the configuration, the interface, the alternate setting, and the endpoint of the USB pipe to be used for a USB bulk or interrupt transfer.*
- struct [UsbCtrl](#)  
*Type specifying the request type encoding the transfer direction, the value and the index of a USB control request.*
- class [UsbDevice](#)  
*Class defining the C++ API for a libusb controlled USB device inheriting the generic [stm::Device](#) C++ API.*
- class [W32Device](#)  
*Class defining the C++ API for a Windows device inheriting the generic [stm::Device](#) C++ API.*

#### Namespaces

- namespace [local](#)

#### Functions

- `std::ostream & operator<< (std::ostream &os, const Device &device)`  
*Insert a description of the Device device into os.*

### 10.2 stm::local Namespace Reference

#### Classes

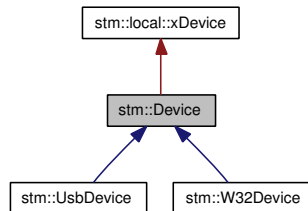
- class [xDevice](#)
- class [xUsbDevice](#)
- class [xW32Device](#)
- class [UsbHandle](#)

## 11 SysToMath IO C++ Libraries Implementation Class Documentation

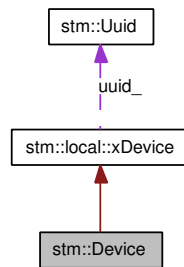
### 11.1 stm::Device Class Reference

```
#include <device.hpp>
```

Inheritance diagram for `stm::Device`:



Collaboration diagram for `stm::Device`:



#### 11.1.1 Detailed Description

Abstract base class defining the C++ API for a generic device.

This class is intended to serve as base class of a special device class which shall implement the interface of [stm::Device](#).

Definition at line 398 of file `device.hpp`.

#### Proxy Implementation

The private member functions `proxy()` may be overwritten in derived classes which shall optionally augment the `stm::Device` interface and implement it.

If this is the case and they return a non-null pointer, this shall be the address of an instance of that implementation class. All `stm::Device` member functions then forward their functionality to that implementation object.

- virtual `Device * proxy ()`
- virtual `const Device * proxy () const`

*The default implementation returns NULL.*

**Public Types**

- enum `OpenMode` {  
`NoAccess` = 0x00000000,  
`ReadAccess` = 0x00000001,  
`WriteAccess` = 0x00000002,  
`ReadWriteAccess` = `ReadAccess` | `WriteAccess` }  
*Open mode flags (bitwise orable).*
- enum `ErrorState` {  
`NoError` = 0,  
`ReadError` = 1,  
`WriteError` = 2,  
`ControlError` = 3,  
`ResourceError` = 4,  
`OpenError` = 5,  
`CloseError` = 6,  
`SeekError` = 7,  
`ArgumentError` = 8,  
`UnknownError` = 9 }  
*Error state values.*
- enum `ErrorFlags` {  
`ErrorFlagMask` = 0x7fff0000,  
`SystemError` = 0x00010000 }  
*Error flags.*
- enum {  
`NoFlags` = 0x00000000,  
`AcceptTimeout` = 0x00000001 }  
*Bitwise orable operation flag bits for `read()`, `write()` and `control()`.*
- enum `DescribeFlags` {  
`IndentMask` = 0x0000000f,  
`IndentFirst` = `IndentMask` + 1,  
`NoPropertyNames` = `IndentFirst` << 1,  
`DefaultProperties` = `NoPropertyNames` << 1,  
`VerboseProperties` = `DefaultProperties` << 1 | `DefaultProperties`,  
`AllProperties` = ~ ((`DefaultProperties` << 2) - 1),  
`DeviceType` = `DefaultProperties` << 2,  
`DeviceUuid` = `DeviceType` << 1,  
`DriverVersion` = `DeviceUuid` << 1 }  
*Describe flags (bitwise orable).*

- enum `Timeout` {  
`DefaultTimeout = -1,`  
`Forever = INT_MAX` }  
*Special timeout values.*

### Public Member Functions

- `Device` (const `Uuid` &uuid=`Uuid`(), const std::string type=std::string(), int defaultTimeout=`Forever`, const `Descriptor` &descr=`Descriptor`())  
*Constructor optionally setting the `Uuid` of the `Device` to uuid, the type string of the `Device` to type, the default operation timeout of the `Device` to defaultTimeout milliseconds and the `Descriptor` of the `Device` to descr.*
- `Device` (int defaultTimeout, const `Descriptor` &descr=`Descriptor`())  
*Constructor setting the default operation timeout of the `Device` to defaultTimeout milliseconds and the `Descriptor` to descr.*
- `Device` (const `Uuid` &uuid, const std::string type, const `Version` &version, int defaultTimeout=`Forever`, const `Descriptor` &descr=`Descriptor`())  
*Constructor setting the `Uuid` of the `Device` to uuid, the type string and version of the `Device` to type and version, respectively, and the default operation timeout of the `Device` to defaultTimeout milliseconds which defaults to infinite.*
- `Device` (const `Device` &other)  
*Copy constructor.*
- `Device` & operator= (const `Device` &other)  
*Copy assignment operator.*
- virtual `~Device` ()  
*Destructor.*
- virtual const `Uuid` & uuid () const  
*The virtual method shall return the `Uuid` of this `Device`.*
- virtual void `setUuid` (const `Uuid` &uuid)  
*The virtual method shall set the `Uuid` of this `Device` to uuid.*
- const std::string & `property` (const std::string &name) const  
*The method returns the string value of property name of this `Device`.*
- void `setProperty` (const std::string &name, const std::string &value)  
*The method sets the property name of this `Device` to value.*
- bool `unsetProperty` (const std::string &name)  
*The method unsets the property name of this `Device` to value and returns true, if it were set, else false.*
- bool `hasProperty` (const std::string &name)  
*The method returns true, if the property name is set for this `Device`, else false.*

- virtual const std::string & **type** () const  
*The virtual method shall return the type string of this [Device](#).*
- virtual void **setType** (const std::string &type)  
*The virtual method shall set the type string of this [Device](#) to type.*
- virtual [Version](#) **version** () const  
*The virtual method shall return the [Version](#) of this [Device](#).*
- virtual void **setVersion** (const [Version](#) &version)  
*The virtual method shall set the [Version](#) of this [Device](#) to version.*
- virtual int **defaultTimeout** () const  
*The virtual method shall return the default timeout of this [Device](#).*
- virtual void **setDefaultTimeout** (int defaultTimeout=Forever)  
*The virtual method shall set the default operation timeout in milliseconds of this [Device](#) to defaultTimeout.*
- virtual const [Descriptor](#) & **descr** () const  
*Return the [Device::Descriptor](#) of this [Device](#).*
- virtual void **setDescr** (const [Descriptor](#) &descr)=0  
*Set the [Device::Descriptor](#) of this [Device](#) to descr.*
- virtual bool **canRead** () const  
*Return the read capability of this [Device](#).*
- virtual bool **canWrite** () const  
*Return the write capability of this [Device](#).*
- virtual bool **canControl** () const  
*Return the control capability of this [Device](#).*
- virtual bool **canSeek** () const  
*Return the seek capability of this [Device](#).*
- virtual int **error** () const  
*Return the error state of this [Device](#).*
- virtual void **setError** (int error, const std::string &msg=std::string()) const  
*Set the error state and error string of this [Device](#) according to error and msg.*
- virtual void **clearError** () const  
*Clear the error state and error string of this [Device](#).*
- virtual std::string **errorString** (bool msgOnly=false) const  
*Return the error string of this [Device](#).*

- virtual void `augmentErrorString` (const std::string &prefix, const std::string &suffix=std::string()) const  
*Augment the error string of this device.*
- virtual bool `isOpen` () const  
*The virtual method shall return true, if this `Device` is open, else false.*
- virtual unsigned int `openMode` () const  
*The virtual method shall return the open mode of this `Device`.*
- virtual bool `open` (unsigned int openMode=ReadWriteAccess)  
*The virtual method shall open this `Device` in openMode, if possible and return true on success, else false.*
- virtual bool `close` ()  
*The virtual method shall close this `Device`, if possible and return true on success, else false.*
- virtual int64\_t `read` (void \*data, int64\_t maxlen, int timeout=DefaultTimeout, unsigned int flags=NoFlags)  
*The virtual method shall read maximal maxlen bytes from this `Device` and store them in the buffer pointed to by data.*
- virtual int64\_t `write` (const void \*data, int64\_t len, int timeout=DefaultTimeout, unsigned int flags=NoFlags)  
*The virtual method shall write len bytes from the buffer pointed to by data to this `Device`.*
- virtual int64\_t `control` (unsigned int request, const void \*inData, int64\_t inLen, void \*outData, int64\_t maxOutLen, int timeout=DefaultTimeout, unsigned int flags=NoFlags) const  
*The virtual method shall perform the control operation specified by request for this `Device` with input data of length inLen pointed to by inData producing output data of maximal length maxOutLen in the buffer pointed to by outData.*
- virtual int64\_t `size` () const  
*The virtual method shall return the size of this `Device`, if it is an opened random access device, else -1.*
- virtual int64\_t `pos` () const  
*The virtual method shall return the access position of this `Device`, if it is an opened random access device, else -1.*
- virtual bool `seek` (int64\_t pos)  
*The virtual method shall set the access position of this `Device` to pos and return true on success, if it is an opened random access device, else it shall return false.*
- virtual bool `reset` ()  
*The virtual method shall set the access position of this `Device` to 0 and return true on success, if it is an opened random access device, else it shall return false.*
- virtual std::ostream & `describe` (std::ostream &os, unsigned int flags=DefaultProperties) const  
*Insert a description of this `Device` into os.*

## Classes

- struct [Descriptor](#)

Objects of type *Device::Descriptor* describe system dependent aspects of a *Device* as a pair of a void pointer and a void function pointer.

- struct [Version](#)

*Device* driver version.

## 11.1.2 Member Enumeration Documentation

### 11.1.2.1 enum stm::Device::OpenMode

Open mode flags (bitwise orable).

The enumerators of [Device::OpenMode](#) specify the possible access modes of a [Device](#).

#### Enumerator:

*NoAccess* No acces, [Device](#) not open.

*ReadAccess* Read access, [Device](#) readable.

*WriteAccess* Write access, [Device](#) writeable.

*ReadWriteAccess* Read and write access.

Definition at line 405 of file device.hpp.

### 11.1.2.2 enum stm::Device::ErrorState

Error state values.

The enumerators of [Device::ErrorState](#) indicate the reason of the last [Device](#) error occurred.

#### Enumerator:

*NoError* No error.

*ReadError* Error during [read\(\)](#).

*WriteError* Error during [write\(\)](#).

*ControlError* Error during [control\(\)](#).

*ResourceError* Resource error.

*OpenError* Error during [open\(\)](#).

*CloseError* Error during [close\(\)](#).

*SeekError* Error during [seek\(\)](#), [reset\(\)](#), [pos\(\)](#) or [size\(\)](#).

*ArgumentError* Invalid argument.

*UnknownError* Unknown error.

Definition at line 416 of file device.hpp.

### 11.1.2.3 `enum stm::Device::ErrorFlags`

Error flags.

#### Enumerator:

*ErrorFlagMask* All error flags shall be covered by that mask.

*SystemError* Indicates that the error string shall be augmented by a system error description, if available.

Definition at line 432 of file `device.hpp`.

### 11.1.2.4 anonymous enum

Bitwise orable operation flag bits for `read()`, `write()` and `control()`.

#### Enumerator:

*NoFlags* No flag bits set.

*AcceptTimeout* Timeout is no error.

Definition at line 442 of file `device.hpp`.

### 11.1.2.5 `enum stm::Device::DescribeFlags`

Describe flags (bitwise orable).

The enumerators of `Device::DescribeFlags` specify output format and extent produced by `describe()`.

#### Enumerator:

*IndentMask* Mask for indentation field.

If the value masked is not 0, each property is output on a new line indented by this value. If it is 0, all properties are enumerated on one line separated by commas.

*IndentFirst* If set, also the first property is indented, else not and also its name is omitted.

This flag has no effect, if no indentation is performed.

*NoPropertyNames* If set and no indentation is performed, all property names are omitted.

*DefaultProperties* If set, the device specific default properties are included.

*VerboseProperties* If set, the device specific verbose properties are included which include the device specific default properties.

*AllProperties* If set, all device specific properties are included.

*DeviceType* If set, the device type property is included, else not.

*DeviceUuid* If set, the device UUID property is included, else not.

*DriverVersion* If set, the driver version property is included, else not.

Reimplemented in `stm::UsbDevice`, and `stm::W32Device`.

Definition at line 451 of file `device.hpp`.

### 11.1.2.6 enum stm::Device::Timeout

Special timeout values.

#### Enumerator:

**DefaultTimeout** Use default timeout of this Device.

**Forever** Wait forever.

Definition at line 489 of file device.hpp.

### 11.1.3 Constructor & Destructor Documentation

**11.1.3.1 stm::Device::Device (const Uuid & uuid = Uuid (), const std::string type = std::string (), int defaultTimeout = Forever, const Descriptor & descr = Descriptor ()) [inline]**

Constructor optionally setting the [Uuid](#) of the [Device](#) to *uuid*, the type string of the [Device](#) to *type*, the default operation timeout of the [Device](#) to *defaultTimeout* milliseconds and the [Descriptor](#) of the [Device](#) to *descr*.

The default [Uuid](#) is null as is the default type string, whereas the default *defaultTimeout* is infinite and the default *descr* is invalid.

Definition at line 475 of file xdevice.hpp.

**11.1.3.2 stm::Device::Device (int defaultTimeout, const Descriptor & descr = Descriptor ()) [inline]**

Constructor setting the default operation timeout of the [Device](#) to *defaultTimeout* milliseconds and the [Descriptor](#) to *descr*.

Definition at line 486 of file xdevice.hpp.

**11.1.3.3 stm::Device::Device (const Uuid & uuid, const std::string type, const Version & version, int defaultTimeout = Forever, const Descriptor & descr = Descriptor ()) [inline]**

Constructor setting the [Uuid](#) of the [Device](#) to *uuid*, the type string and version of the [Device](#) to *type* and *version*, respectively, and the default operation timeout of the [Device](#) to *defaultTimeout* milliseconds which defaults to infinite.

Moreover, the Descriptor is set to *descr* defaulting to invalid.

Definition at line 495 of file xdevice.hpp.

**11.1.3.4 stm::Device::Device (const Device & other) [inline]**

Copy constructor.

Only copyable, if the [Device::Descriptor](#) of *other* is invalid. Then the constructed [Device](#) object is a copy of *other* with the exception that its [Uuid](#) is null.

Definition at line 506 of file xdevice.hpp.

References [descr\(\)](#).

### 11.1.3.5 stm::Device::~~Device () [inline, virtual]

Destructor.

If this [Device](#) is open, it is closed first.

Definition at line 537 of file `xdevice.hpp`.

References `close()`, and `isOpen()`.

## 11.1.4 Member Function Documentation

### 11.1.4.1 Device & stm::Device::operator= (const Device & other) [inline]

Copy assignment operator.

Only copyable, if the [Device::Descriptor](#) of this [Device](#) and of *other* are invalid. Then this [Device](#) is replaced with a copy of *other* with the exception that its [Uuid](#) is null.

Definition at line 523 of file `xdevice.hpp`.

References `defaultTimeout()`, `descr()`, `setDefaultTimeout()`, `setType()`, `setUuid()`, `setVersion()`, `type()`, and `version()`.

### 11.1.4.2 const Uuid & stm::Device::uuid () const [inline, virtual]

The virtual method shall return the [Uuid](#) of this [Device](#).

The default implementation returns the [Uuid](#) of this [Device](#) set by the constructor or by `setUuid()`.

Definition at line 546 of file `xdevice.hpp`.

References `proxy()`, and `stm::local::xDevice::uuid_`.

Referenced by `describe()`.

### 11.1.4.3 void stm::Device::setUuid (const Uuid & uuid) [inline, virtual]

The virtual method shall set the [Uuid](#) of this [Device](#) to *uuid*.

The default implementation sets the [Uuid](#) of this device to *uuid*.

Definition at line 556 of file `xdevice.hpp`.

References `proxy()`, and `stm::local::xDevice::uuid_`.

Referenced by `operator=()`.

### 11.1.4.4 const std::string & stm::Device::property (const std::string & name) const [inline]

The method returns the string value of property *name* of this [Device](#).

If property *name* was not set, an empty string is returned.

Definition at line 568 of file `xdevice.hpp`.

References `stm::local::xDevice::property_`, and `proxy()`.

Referenced by `close()`, `stm::W32Device::close()`, `control()`, `stm::W32Device::control()`, `stm::UsbDevice::control()`, `open()`, `stm::W32Device::open()`, `stm::UsbDevice::open()`, `pos()`, `read()`, `stm::W32Device::read()`, `stm::UsbDevice::read()`, `stm::UsbDevice::readPipe()`, `seek()`, `size()`, `type()`, `write()`, `stm::W32Device::write()`, `stm::UsbDevice::write()`, and `stm::UsbDevice::writePipe()`.

**11.1.4.5** `void stm::Device::setProperty (const std::string & name, const std::string & value)` [inline]

The method sets the property *name* of this [Device](#) to *value*.

Definition at line 580 of file `xdevice.hpp`.

References `stm::local::xDevice::property_`, and `proxy()`.

Referenced by `setType()`.

**11.1.4.6** `bool stm::Device::unsetProperty (const std::string & name)` [inline]

The method unsets the property *name* of this [Device](#) to *value* and returns true, if it were set, else false.

Definition at line 591 of file `xdevice.hpp`.

References `stm::local::xDevice::property_`, and `proxy()`.

**11.1.4.7** `bool stm::Device::hasProperty (const std::string & name)` [inline]

The method returns true, if the property *name* is set for this [Device](#), else false.

Definition at line 601 of file `xdevice.hpp`.

References `stm::local::xDevice::property_`, and `proxy()`.

**11.1.4.8** `const std::string & stm::Device::type () const` [inline, virtual]

The virtual method shall return the type string of this [Device](#).

The default implementation returns the type string of this [Device](#) set by the constructor or by `setType()` and is implemented as property.

Definition at line 611 of file `xdevice.hpp`.

References `property()`.

Referenced by `describe()`, and `operator=()`.

**11.1.4.9** `void stm::Device::setType (const std::string & type)` [inline, virtual]

The virtual method shall set the type string of this [Device](#) to *type*.

The default implementation sets the type string of this device to *type* and is implemented as property.

Definition at line 617 of file `xdevice.hpp`.

References `setProperty()`.

Referenced by `operator=()`.

**11.1.4.10** `Device::Version stm::Device::version () const` [inline, virtual]

The virtual method shall return the [Version](#) of this [Device](#).

The default implementation returns the [Version](#) of this [Device](#) set by the constructor or by `setVersion()`.

Definition at line 624 of file `xdevice.hpp`.

References `stm::Device::Version::part`, `proxy()`, and `stm::local::xDevice::version_`.

Referenced by `describe()`, and `operator=()`.

**11.1.4.11 void stm::Device::setVersion (const Version & version) [inline, virtual]**

The virtual method shall set the [Version](#) of this [Device](#) to *version*.

The default implementation sets the [Version](#) of this device to *version*.

Definition at line 636 of file xdevice.hpp.

References `stm::Device::Version::part`, `proxy()`, and `stm::local::xDevice::version_`.

Referenced by `operator=()`.

**11.1.4.12 int stm::Device::defaultTimeout () const [inline, virtual]**

The virtual method shall return the default timeout of this [Device](#).

The default implementation returns the default operation timeout of this [Device](#) in milliseconds set by the constructor or by `setDefaultTimeout()`.

Definition at line 648 of file xdevice.hpp.

References `stm::local::xDevice::defaultTimeout_`, and `proxy()`.

Referenced by `stm::W32Device::control()`, `stm::UsbDevice::control()`, `operator=()`, `stm::W32Device::read()`, `stm::UsbDevice::read()`, `stm::W32Device::write()`, and `stm::UsbDevice::write()`.

**11.1.4.13 void stm::Device::setDefaultTimeout (int defaultTimeout = Forever) [inline, virtual]**

The virtual method shall set the default operation timeout in milliseconds of this [Device](#) to *defaultTimeout*.

The default implementation sets the default timeout of this [Device](#) to *defaultTimeout* which defaults to infinite.

Definition at line 658 of file xdevice.hpp.

References `stm::local::xDevice::defaultTimeout_`, `Forever`, and `proxy()`.

Referenced by `operator=()`.

**11.1.4.14 const Device::Descriptor & stm::Device::descr () const [inline, virtual]**

Return the [Device::Descriptor](#) of this [Device](#).

**Returns:**

An invalid [Device::Descriptor](#), if this [Device](#) does not represent a device.

A valid [Device::Descriptor](#) of this [Device](#) represents a device.

**Note:**

It is not necessary that this [UsbDevice](#) is open.

**See also:**

[setDescr\(\)](#).

Definition at line 673 of file xdevice.hpp.

References `stm::local::xDevice::descr_`.

Referenced by `stm::W32Device::describe()`, `Device()`, `stm::W32Device::isA()`, `stm::UsbDevice::isA()`, `stm::W32Device::open()`, `operator=()`, `stm::UsbDevice::setDescr()`, and `stm::UsbDevice::setPipe()`.

**11.1.4.15** `void stm::Device::setDescr (const Descriptor & descr)` [`inline`, `pure virtual`]

Set the `Device::Descriptor` of this `Device` to `descr`.

**Parameters:**

← `descr` A valid `Device::Descriptor` of the device to be represented by this `Device` object or an invalid `Device::Descriptor`.

**Effects:**

If this `Device` is open, it is closed. Then `descr` is defined for it. If `descr` is valid, this `Device` is ready to be opened, else it does not represent a device.

**See also:**

`descr()`, `isOpen()`, `close()`, `open()`.

Implemented in `stm::UsbDevice`.

Definition at line 679 of file `xdevice.hpp`.

References `close()`, `stm::local::xDevice::descr_`, and `isOpen()`.

Referenced by `stm::UsbDevice::setDescr()`.

**11.1.4.16** `bool stm::Device::canRead () const` [`inline`, `virtual`]

Return the read capability of this `Device`.

**Returns:**

`false`.

**Note:**

If this virtual method is not reimplemented by a derived class, this means that the device cannot be successfully opened in open mode `Device::ReadAccess`.

It is not necessary that this `Device` is open.

**See also:**

`canWrite()`, `canControl()`, `canSeek()`, `open()`.

Reimplemented in `stm::UsbDevice`, and `stm::W32Device`.

Definition at line 690 of file `xdevice.hpp`.

References `proxy()`.

Referenced by `open()`.

**11.1.4.17** `bool stm::Device::canWrite () const` [`inline`, `virtual`]

Return the write capability of this `Device`.

**Returns:**

`false`.

**Note:**

If this virtual method is not reimplemented by a derived class, this means that the device cannot be successfully opened in open mode `Device::WriteAccess`.  
It is not necessary that this `Device` is open.

**See also:**

[canRead\(\)](#), [canControl\(\)](#), [canSeek\(\)](#), [open\(\)](#).

Reimplemented in [stm::UsbDevice](#), and [stm::W32Device](#).

Definition at line 700 of file `xdevice.hpp`.

References [proxy\(\)](#).

Referenced by [open\(\)](#).

**11.1.4.18** `bool stm::Device::canControl () const` [`inline`, `virtual`]

Return the control capability of this `Device`.

**Returns:**

`false`.

**Note:**

If this virtual method is not reimplemented by a derived class, this means that the device does not support the method [control\(\)](#).  
It is not necessary that this `Device` is open.

**See also:**

[canRead\(\)](#), [canWrite\(\)](#), [canSeek\(\)](#), [control\(\)](#).

Reimplemented in [stm::UsbDevice](#), and [stm::W32Device](#).

Definition at line 710 of file `xdevice.hpp`.

References [proxy\(\)](#).

**11.1.4.19** `bool stm::Device::canSeek () const` [`inline`, `virtual`]

Return the seek capability of this `Device`.

**Returns:**

`false`.

**Note:**

If this virtual method is not reimplemented by a derived class, this means that the device does not support the methods [size\(\)](#), [pos\(\)](#), [seek \(\)](#) and [reset\(\)](#).  
It is not necessary that this `Device` is open.

**See also:**

[canRead\(\)](#), [canWrite\(\)](#), [canControl\(\)](#), [size\(\)](#), [pos\(\)](#), [seek\(\)](#), [reset \(\)](#).

Definition at line 720 of file `xdevice.hpp`.

References `proxy()`.

Referenced by `pos()`, `seek()`, and `size()`.

#### 11.1.4.20 `int stm::Device::error () const` [`inline`, `virtual`]

Return the error state of this `Device`.

##### Returns:

The error state of this `Device` as one of the enumerators of `Device::ErrorState`.

##### Note:

It is not necessary that this `Device` is open.

##### See also:

`setError()`, `clearError()`, `errorString()`, `augmentErrorString()`.

Reimplemented in `stm::W32Device`.

Definition at line 730 of file `xdevice.hpp`.

References `stm::local::xDevice::errMsg_`, `stm::local::xDevice::error_`, `NoError`, and `proxy()`.

Referenced by `stm::W32Device::error()`.

#### 11.1.4.21 `void stm::Device::setError (int error, const std::string & msg = std::string ()) const` [`inline`, `virtual`]

Set the error state and error string of this `Device` according to `error` and `msg`.

##### Parameters:

← `error` Error state as one of the enumerators of `Device::ErrorState` optionally ored with one ore more of the enumerators of `Device::ErrorFlags`.

← `msg` Error string.

##### Effects:

If the error state part of `error` is one of the enumerators of `Device::ErrorState`, the error state of this `Device` is set to that state and its error string to `msg`, else to `Device::UnknownError`. If the error flag `Device::SystemError` is set in `error`, the error string is augmented by a system error description, if available.

##### Note:

Despite of being `const`, the method can change the error state and error string.  
It is not necessary that this `Device` is open.

##### See also:

`error()`, `clearError()`, `errorString()`, `augmentErrorString()`.

Reimplemented in `stm::UsbDevice`, and `stm::W32Device`.

Definition at line 744 of file xdevice.hpp.

References `stm::local::xDevice::errMsg_`, `stm::local::xDevice::error_`, `ErrorFlagMask`, `NoError`, `proxy()`, and `UnknownError`.

Referenced by `close()`, `control()`, `open()`, `pos()`, `read()`, `seek()`, `stm::W32Device::setError()`, `stm::UsbDevice::setError()`, `size()`, and `write()`.

#### 11.1.4.22 void stm::Device::clearError () const [inline, virtual]

Clear the error state and error string of this [Device](#).

##### Effects:

The error state and error string of this [Device](#) are cleared, that means set to [Device::NoError](#) and the empty string.

##### Note:

Despite of being `const`, the method can change the error state and error string. It is not necessary that this [Device](#) is open.

##### See also:

[error\(\)](#), [setError\(\)](#), [errorString\(\)](#), [augmentErrorString\(\)](#).

Reimplemented in [stm::W32Device](#).

Definition at line 761 of file xdevice.hpp.

References `stm::local::xDevice::errMsg_`, `stm::local::xDevice::error_`, `NoError`, and `proxy()`.

Referenced by `stm::W32Device::clearError()`.

#### 11.1.4.23 std::string stm::Device::errorString (bool msgOnly = false) const [inline, virtual]

Return the error string of this [Device](#).

##### Parameters:

← *msgOnly* If true, return only error message, else precede it by a verbal description of the error state.

##### Returns:

A non empty string describing the error state of this [Device](#), if that error state is not [Device::NoError](#). The empty string, if the error state of this [Device](#) is [Device::NoError](#).

##### Note:

It is not necessary that this [Device](#) is open.

##### See also:

[error\(\)](#), [setError\(\)](#), [clearError\(\)](#), [augmentErrorString\(\)](#).

Definition at line 774 of file xdevice.hpp.

References `stm::local::xDevice::errMsg_`, `stm::local::xDevice::error_`, and `proxy()`.

**11.1.4.24** `void stm::Device::augmentErrorString (const std::string & prefix, const std::string & suffix = std::string ()) const` [inline, virtual]

Augment the error string of this device.

**Parameters:**

← *prefix* Error string prefix.

← *suffix* Error string suffix.

**Effects:**

If the error state of this [Device](#) is not [Device::NoError](#) and at least one of *prefix* or *suffix* is not empty, the current error string is augmented accordingly.

**Note:**

Despite of being const, the method can change the error string.  
It is not necessary that this [Device](#) is open.

**See also:**

[error\(\)](#), [setError\(\)](#), [clearError\(\)](#), [errorString\(\)](#).

Definition at line 803 of file `xdevice.hpp`.

References `stm::local::xDevice::errMsg_`, `stm::local::xDevice::error_`, `NoError`, and `proxy()`.

**11.1.4.25** `bool stm::Device::isOpen () const` [inline, virtual]

The virtual method shall return true, if this [Device](#) is open, else false.

The default implementation returns true, if this [Device](#) is open, that is if its open mode is not the [Device::OpenMode](#) enumerator [Device::NoAccess](#), else false.

Reimplemented in [stm::UsbDevice](#), and [stm::W32Device](#).

Definition at line 842 of file `xdevice.hpp`.

References `NoAccess`, `stm::local::xDevice::openMode_`, and `proxy()`.

Referenced by `close()`, `control()`, `stm::W32Device::isOpen()`, `stm::UsbDevice::isOpen()`, `open()`, `setDescription()`, and `~Device()`.

**11.1.4.26** `unsigned int stm::Device::openMode () const` [inline, virtual]

The virtual method shall return the open mode of this [Device](#).

The default implementation returns the open mode of this [Device](#) as one of the enumerators of [Device::OpenMode](#).

Definition at line 852 of file `xdevice.hpp`.

References `stm::local::xDevice::openMode_`, and `proxy()`.

Referenced by `read()`, and `write()`.

**11.1.4.27** `bool stm::Device::open (unsigned int openMode = ReadWriteAccess)` [inline, virtual]

The virtual method shall open this [Device](#) in *openMode*, if possible and return true on success, else false.

The default implementation sets the error state of this [Device](#) to the [Device::ErrorState](#) enumerator [Device::OpenError](#) and returns false, if [isOpen\(\)](#) does not return false or if *openMode* is not compatible with the results of [canRead\(\)](#) and/or [canWrite\(\)](#). Else the method sets the open mode of this [Device](#) to *openMode* and returns true.

Reimplemented in [stm::UsbDevice](#), and [stm::W32Device](#).

Definition at line 862 of file [xdevice.hpp](#).

References [canRead\(\)](#), [canWrite\(\)](#), [isOpen\(\)](#), [NoAccess](#), [OpenError](#), [stm::local::xDevice::openMode\\_](#), [property\(\)](#), [proxy\(\)](#), [ReadAccess](#), [setError\(\)](#), and [WriteAccess](#).

Referenced by [stm::W32Device::open\(\)](#), and [stm::UsbDevice::open\(\)](#).

#### 11.1.4.28 bool stm::Device::close () [inline, virtual]

The virtual method shall close this [Device](#), if possible and return true on success, else false.

The default implementation sets the error state of this [Device](#) to the [Device::ErrorState](#) enumerator [Device::CloseError](#) and returns false, if [isOpen\(\)](#) returns false. Else the method sets the open mode of this [Device](#) to the [Device::OpenMode](#) enumerator [Device::NoAccess](#) and returns true.

Reimplemented in [stm::UsbDevice](#), and [stm::W32Device](#).

Definition at line 899 of file [xdevice.hpp](#).

References [CloseError](#), [isOpen\(\)](#), [NoAccess](#), [stm::local::xDevice::openMode\\_](#), [property\(\)](#), [proxy\(\)](#), and [setError\(\)](#).

Referenced by [stm::W32Device::close\(\)](#), [stm::UsbDevice::close\(\)](#), [stm::W32Device::open\(\)](#), [stm::UsbDevice::open\(\)](#), [setDescr\(\)](#), and [~Device\(\)](#).

#### 11.1.4.29 int64\_t stm::Device::read (void \* data, int64\_t maxlen, int timeout = DefaultTimeout, unsigned int flags = NoFlags) [inline, virtual]

The virtual method shall read maximal *maxLen* bytes from this [Device](#) and store them in the buffer pointed to by *data*.

On error the method shall return -1, else the number of bytes actually read. The default implementation sets the error state of this [Device](#) to [Device::ErrorState](#) enumerator [Device::ReadError](#) and returns -1, if the result of [openMode\(\)](#) does not contain the [Device::OpenMode](#) enumerator [Device::ReadAccess](#) or if *maxLen* is negative or *data* is the NULL pointer unless *maxLen* is also 0. Else the method returns *maxLen*.

Reimplemented in [stm::UsbDevice](#), and [stm::W32Device](#).

Definition at line 930 of file [xdevice.hpp](#).

References [DefaultTimeout](#), [openMode\(\)](#), [property\(\)](#), [proxy\(\)](#), [ReadAccess](#), [ReadError](#), and [setError\(\)](#).

Referenced by [stm::W32Device::read\(\)](#), and [stm::UsbDevice::read\(\)](#).

#### 11.1.4.30 int64\_t stm::Device::write (const void \* data, int64\_t len, int timeout = DefaultTimeout, unsigned int flags = NoFlags) [inline, virtual]

The virtual method shall write *len* bytes from the buffer pointed to by *data* to this [Device](#).

On error the method shall return -1, else the number of bytes actually written. The default implementation sets the error state of this [Device](#) to the [Device::ErrorState](#) enumerator [Device::WriteError](#) and returns -1, if the result of [openMode\(\)](#) does not contain the [Device::OpenMode](#) enumerator [Device::WriteAccess](#) or if *len* is negative or *data* is the NULL pointer unless *len* is also 0. Else the method returns *len*.

Reimplemented in [stm::UsbDevice](#), and [stm::W32Device](#).

Definition at line 969 of file `xdevice.hpp`.

References `DefaultTimeout`, `openMode()`, `property()`, `proxy()`, `setError()`, `WriteAccess`, and `WriteError`.

Referenced by `stm::W32Device::write()`, and `stm::UsbDevice::write()`.

**11.1.4.31** `int64_t stm::Device::control (unsigned int request, const void * inData, int64_t inLen, void * outData, int64_t maxOutLen, int timeout = DefaultTimeout, unsigned int flags = NoFlags) const` `[inline, virtual]`

The virtual method shall perform the control operation specified by *request* for this [Device](#) with input data of length *inLen* pointed to by *inData* producing output data of maximal length *maxOutLen* in the buffer pointed to by *outData*.

On error the method shall return -1, else the number of bytes produced in *outData*. The default implementation sets the error state of this [Device](#) to the [Device::ErrorState](#) enumerator [Device::ControlError](#) and returns -1, if `canControl()` returns false, or if *inLen* is negative or *inData* is NULL unless *inLen* is also 0, or if *outLen* is negative or *outData* is NULL unless *maxOutLen* is also 0. Else the method returns *maxOutLen*.

Reimplemented in [stm::UsbDevice](#), and [stm::W32Device](#).

Definition at line 1008 of file `xdevice.hpp`.

References `ControlError`, `DefaultTimeout`, `isOpen()`, `property()`, `proxy()`, and `setError()`.

Referenced by `stm::W32Device::control()`, and `stm::UsbDevice::control()`.

**11.1.4.32** `int64_t stm::Device::size () const` `[inline, virtual]`

The virtual method shall return the size of this [Device](#), if it is an opened random access device, else -1.

The default implementation sets the error state of this [Device](#) to the [Device::ErrorState](#) enumerator [Device::SeekError](#) and returns -1, if `canSeek()` or `isOpen()` return false. Else the method returns 0.

Definition at line 1058 of file `xdevice.hpp`.

References `canSeek()`, `property()`, `proxy()`, `SeekError`, and `setError()`.

**11.1.4.33** `int64_t stm::Device::pos () const` `[inline, virtual]`

The virtual method shall return the access position of this [Device](#), if it is an opened random access device, else -1.

The default implementation sets the error state of this [Device](#) to the [Device::ErrorState](#) enumerator [Device::SeekError](#) and returns -1, if `canSeek()` or `isOpen()` return false. Else the method returns 0.

Definition at line 1079 of file `xdevice.hpp`.

References `canSeek()`, `property()`, `proxy()`, `SeekError`, and `setError()`.

**11.1.4.34** `bool stm::Device::seek (int64_t pos)` `[inline, virtual]`

The virtual method shall set the access position of this [Device](#) to *pos* and return true on success, if it is an opened random access device, else it shall return false.

The default implementation sets the error state of this [Device](#) to the [Device::ErrorState](#) enumerator [Device::SeekError](#) and returns false, if `canSeek()` or `isOpen()` return false, or if *pos* is negative. Else the method returns true.

Definition at line 1100 of file `xdevice.hpp`.

References `canSeek()`, `property()`, `proxy()`, `SeekError`, and `setError()`.

Referenced by `reset()`.

#### 11.1.4.35 `bool stm::Device::reset () [inline, virtual]`

The virtual method shall set the access position of this `Device` to 0 and return true on success, if it is an opened random access device, else it shall return false.

The default implementation returns `seek(0)`.

Definition at line 1122 of file `xdevice.hpp`.

References `seek()`.

#### 11.1.4.36 `std::ostream & stm::Device::describe (std::ostream & os, unsigned int flags = DefaultProperties) const [inline, virtual]`

Insert a description of this `Device` into `os`.

The method shall insert a verbal description of this `Device` into the output stream `os` and return `os`. Format and extent of the description shall be controlled by the `flags` parameter according to the bitwise ored enumerators of `Device::DescribeFlags`. The default implementation handles the device properties device type and device `Uuid` for `DefaultProperties` and additionally the driver version property for `VerboseProperties` or `AllProperties`.

Reimplemented in `stm::UsbDevice`, and `stm::W32Device`.

Definition at line 1128 of file `xdevice.hpp`.

References `DefaultProperties`, `DeviceType`, `DeviceUuid`, `DriverVersion`, `IndentFirst`, `IndentMask`, `NoPropertyNames`, `proxy()`, `stm::Device::Version::string()`, `stm::Uuid::string()`, `type()`, `uuid()`, `VerboseProperties`, and `version()`.

Referenced by `stm::UsbDevice::describe()`, `stm::W32Device::describe()`, and `stm::operator<<()`.

#### 11.1.4.37 `Device * stm::Device::proxy () [inline, private, virtual]`

The default implementation returns `NULL`.

Definition at line 463 of file `xdevice.hpp`.

Referenced by `augmentErrorString()`, `canControl()`, `canRead()`, `canSeek()`, `canWrite()`, `clearError()`, `close()`, `control()`, `defaultTimeout()`, `describe()`, `error()`, `errorString()`, `hasProperty()`, `isOpen()`, `open()`, `openMode()`, `pos()`, `property()`, `read()`, `seek()`, `setDefaultTimeout()`, `setError()`, `setProperty()`, `setUuid()`, `setVersion()`, `size()`, `unsetProperty()`, `uuid()`, `version()`, and `write()`.

#### 11.1.4.38 `const Device * stm::Device::proxy () const [inline, private, virtual]`

The default implementation returns `NULL`.

Definition at line 469 of file `xdevice.hpp`.

The documentation for this class was generated from the following files:

- [device.hpp](#)
- [xdevice.hpp](#)

## 11.2 `stm::Device::Descriptor` Struct Reference

```
#include <device.hpp>
```

### 11.2.1 Detailed Description

Objects of type `Device::Descriptor` describe system dependent aspects of a `Device` as a pair of a void pointer and a void function pointer.

If a `Device::Descriptor` object's first pointer is NULL the descriptor is called invalid else valid.

Definition at line 502 of file `device.hpp`.

### Public Member Functions

- `Descriptor` (void \**d*=NULL, void(\**f*)()=NULL)  
*Constructs a `Device::Descriptor` object called invalid, if *d* is NULL.*
- `operator const void * () const`  
*Returns the first pointer of the `Descriptor`.*

### 11.2.2 Constructor & Destructor Documentation

#### 11.2.2.1 `stm::Device::Descriptor::Descriptor` (void \* *d* = NULL, void(\**f*)()= NULL) [inline]

Constructs a `Device::Descriptor` object called invalid, if *d* is NULL.

Definition at line 452 of file `xdevice.hpp`.

### 11.2.3 Member Function Documentation

#### 11.2.3.1 `stm::Device::Descriptor::operator const void * () const` [inline]

Returns the first pointer of the `Descriptor`.

Definition at line 457 of file `xdevice.hpp`.

The documentation for this struct was generated from the following files:

- `device.hpp`
- `xdevice.hpp`

## 11.3 `stm::Device::Version` Struct Reference

```
#include <device.hpp>
```

### 11.3.1 Detailed Description

`Device` driver version.

Definition at line 922 of file `device.hpp`.

### Public Types

- enum `Parts` {  
    `Major` = 3,  
    `Minor` = 2,  
    `Micro` = 1,  
    `Nano` = 0 }  
    *Version part names.*

### Public Member Functions

- `Version ()`  
    Construct the null `Version` object with all part words cleared.
- `Version (unsigned short major, unsigned minor, unsigned micro=0, unsigned nano=0)`  
    Construct the `Version` object with all parts major, minor, micro and nano.
- `Version (const Version &other)`  
    Copy constructor.
- `Version & operator= (const Version &other)`  
    Assignment operator.
- `std::string string () const`  
    Return the string representation of this `Version`.
- `bool isNull () const`  
    Returns true, if this `Version` is null.
- `bool operator== (const Version &other) const`  
    Equality comparison operator.
- `bool operator!= (const Version &other) const`  
    Unequality comparison operator.
- `bool operator< (const Version &other) const`  
    Less than comparison operator.
- `bool operator> (const Version &other) const`  
    Greater than comparison operator.
- `bool operator<= (const Version &other) const`  
    Less or equal comparison operator.
- `bool operator>= (const Version &other) const`  
    Greater or equal comparison operator.

## Public Attributes

- unsigned short [part](#) [4]  
*Word array of version parts.*

## 11.3.2 Member Enumeration Documentation

### 11.3.2.1 enum stm::Device::Version::Parts

[Version](#) part names.

They serve as indexes of the part array.

#### Enumerator:

- Major* Major version part.
- Minor* Minor version part.
- Micro* Micro version part.
- Nano* Nano version part.

Definition at line 926 of file device.hpp.

## 11.3.3 Constructor & Destructor Documentation

### 11.3.3.1 stm::Device::Version::Version () [inline]

Construct the null [Version](#) object with all part words cleared.

Definition at line 358 of file xdevice.hpp.

References [part](#).

Referenced by [isNull\(\)](#).

### 11.3.3.2 stm::Device::Version::Version (unsigned short *major*, unsigned *minor*, unsigned *micro* = 0, unsigned *nano* = 0) [inline]

Construct the [Version](#) object with all parts *major*, *minor*, *micro* and *nano*.

Definition at line 365 of file xdevice.hpp.

References [Major](#), [Micro](#), [Minor](#), [Nano](#), and [part](#).

### 11.3.3.3 stm::Device::Version::Version (const Version & *other*) [inline]

Copy constructor.

Definition at line 379 of file xdevice.hpp.

References [part](#).

## 11.3.4 Member Function Documentation

### 11.3.4.1 Device::Version & stm::Device::Version::operator= (const Version & *other*) [inline]

Assignment operator.

Definition at line 385 of file `xdevice.hpp`.

References part.

#### 11.3.4.2 `std::string stm::Device::Version::string () const` [inline]

Return the string representation of this [Version](#).

Definition at line 392 of file `xdevice.hpp`.

References Major, Micro, Minor, Nano, and part.

Referenced by `stm::Device::describe()`.

#### 11.3.4.3 `bool stm::Device::Version::isNull () const` [inline]

Returns true, if this [Version](#) is null.

That means, if all part words are cleared.

Definition at line 403 of file `xdevice.hpp`.

References `Version()`.

#### 11.3.4.4 `bool stm::Device::Version::operator==(const Version & other) const` [inline]

Equality comparison operator.

Definition at line 409 of file `xdevice.hpp`.

References part.

#### 11.3.4.5 `bool stm::Device::Version::operator!=(const Version & other) const` [inline]

Unequality comparison operator.

Definition at line 415 of file `xdevice.hpp`.

References part.

#### 11.3.4.6 `bool stm::Device::Version::operator<(const Version & other) const` [inline]

Less than comparison operator.

Definition at line 421 of file `xdevice.hpp`.

References Major, Micro, Minor, Nano, and part.

#### 11.3.4.7 `bool stm::Device::Version::operator>(const Version & other) const` [inline]

Greater than comparison operator.

Definition at line 434 of file `xdevice.hpp`.

#### 11.3.4.8 `bool stm::Device::Version::operator<=(const Version & other) const` [inline]

Less or equal comparison operator.

Definition at line 440 of file `xdevice.hpp`.

#### 11.3.4.9 bool stm::Device::Version::operator>= (const Version & other) const [inline]

Greater or equal comparison operator.

Definition at line 446 of file xdevice.hpp.

### 11.3.5 Member Data Documentation

#### 11.3.5.1 unsigned short stm::Device::Version::part[4]

Word array of version parts.

Its elements store the version parts indexed by the enumerators of Parts.

Definition at line 976 of file device.hpp.

Referenced by operator!=(), operator<(), operator=(), operator==(), stm::Device::setVersion(), string(), stm::Device::version(), and Version().

The documentation for this struct was generated from the following files:

- [device.hpp](#)
- [xdevice.hpp](#)

## 11.4 stm::UsbCtrl Struct Reference

```
#include <usbdevice.hpp>
```

### 11.4.1 Detailed Description

Type specifying the request type encoding the transfer direction, the value and the index of a USB control request.

Definition at line 162 of file usbdevice.hpp.

#### Public Member Functions

- [UsbCtrl](#) (int *requestType*=0, int *value*=0, int *index*=0)

#### Public Attributes

- int [requestType](#)
- int [value](#)
- int [index](#)

### 11.4.2 Constructor & Destructor Documentation

#### 11.4.2.1 stm::UsbCtrl::UsbCtrl (int *requestType* = 0, int *value* = 0, int *index* = 0) [inline]

Definition at line 283 of file xusbdevice.hpp.

### 11.4.3 Member Data Documentation

#### 11.4.3.1 int stm::UsbCtrl::requestType

Definition at line 171 of file usbdevice.hpp.

#### 11.4.3.2 int stm::UsbCtrl::value

Definition at line 172 of file usbdevice.hpp.

#### 11.4.3.3 int stm::UsbCtrl::index

Definition at line 173 of file usbdevice.hpp.

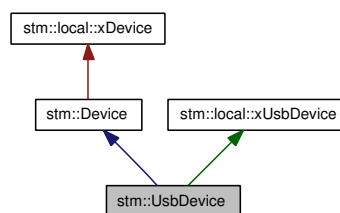
The documentation for this struct was generated from the following files:

- [usbdevice.hpp](#)
- [xusbdevice.hpp](#)

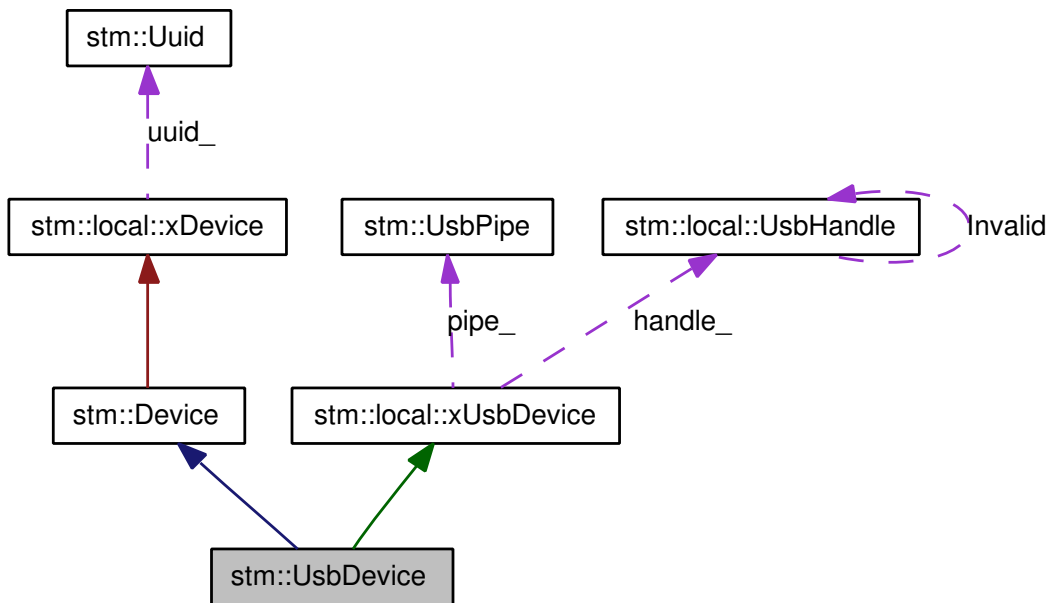
## 11.5 stm::UsbDevice Class Reference

```
#include <usbdevice.hpp>
```

Inheritance diagram for stm::UsbDevice:



Collaboration diagram for stm::UsbDevice:



### 11.5.1 Detailed Description

Class defining the C++ API for a libusb controlled USB device inheriting the generic [stm::Device](#) C++ API.

This class can be instantiated or be used as base class of a special libusb controlled USB device class which may reimplement the interface of [stm::UsbDevice](#) as well as that of the abstract base class [stm::Device](#).

Definition at line 199 of file usbdevice.hpp.

#### Public Types

- enum [DescribeFlags](#) {
  - [DeviceReleaseNumber](#) = DriverVersion << 1,
  - [DeviceBusNumber](#) = DeviceReleaseNumber << 1,
  - [DeviceManufacturer](#) = DeviceBusNumber << 1,
  - [DeviceProduct](#) = DeviceManufacturer << 1,
  - [DeviceSerialNumber](#) = DeviceProduct << 1,
  - [DeviceConfigurations](#) = DeviceSerialNumber << 1,
  - [DeviceInterfaces](#) = DeviceConfigurations << 1,
  - [DeviceAltSettings](#) = DeviceInterfaces << 1,
  - [DeviceEndpoints](#) = DeviceAltSettings << 1,
  - [DeviceChildren](#) = DeviceEndpoints << 1 }

*Describe flags (bitwise orable).*

## Public Member Functions

- **UsbDevice** (int defaultTimeout=Forever, const **Descriptor** &descr=**Descriptor**())  
*Constructor of a **UsbDevice** object representing a libusb controlled USB device described by descr.*
- virtual **~UsbDevice** ()  
*Destructor.*
- virtual bool **canRead** () const  
*Return the read capability of this **UsbDevice**.*
- virtual bool **canWrite** () const  
*Return the write capability of this **UsbDevice**.*
- virtual bool **canControl** () const  
*Return the control capability of this **UsbDevice**.*
- virtual void **setError** (int error, const std::string &msg=std::string()) const  
*Set the error state and error string of this **UsbDevice** according to error and msg.*
- virtual void **setDescr** (const **Descriptor** &descr)  
*Set the **Device::Descriptor** of this **UsbDevice** to descr.*
- **UsbPipe** pipe () const  
*Return a copy of the **UsbPipe** object configured for this **UsbDevice**.*
- **UsbPipe::Type** pipeType () const  
*Return the type of the **UsbPipe** object configured for this **UsbDevice** as an enumerator of the enumeration **UsbPipe::Type**.*
- **UsbPipe::Type** setPipe (**UsbPipe** pipe)  
*Set the **UsbPipe** object configured for this **UsbDevice** to a copy of pipe and return its type.*
- virtual bool **isOpen** () const  
*Determine, if this **UsbDevice** is open.*
- virtual bool **open** (unsigned int openMode)  
*Open this **UsbDevice** in openMode.*
- virtual bool **close** ()  
*Close this **UsbDevice**.*
- virtual int64\_t **read** (void \*data, int64\_t maxLen, int timeout=DefaultTimeout, unsigned int flags=NoFlags)  
*Read maxLen bytes from the currently configured USB pipe of this **UsbDevice** into the data buffer.*
- virtual int64\_t **readPipe** (**UsbPipe** pipe, void \*data, int64\_t maxLen, int timeout=DefaultTimeout, unsigned int flags=NoFlags)  
*Atomically set the **UsbPipe** object configured for this **UsbDevice** to a copy of pipe and read maxLen bytes from that pipe into the data buffer.*

- virtual int64\_t [write](#) (const void \*data, int64\_t len, int timeout=DefaultTimeout, unsigned int flags=NoFlags)  
*Write len bytes from the data buffer to the currently configured USB pipe of this [UsbDevice](#).*
- virtual int64\_t [writePipe](#) ([UsbPipe](#) pipe, const void \*data, int64\_t len, int timeout=DefaultTimeout, unsigned int flags=NoFlags)  
*Atomically set the [UsbPipe](#) object configured for this [UsbDevice](#) to a copy of pipe and write len bytes from the data buffer to that pipe.*
- virtual int64\_t [control](#) (unsigned int request, const void \*ctrl, int64\_t ctrlLen, void \*data, int64\_t dataLen, int timeout=DefaultTimeout, unsigned int flags=NoFlags) const  
*Perform the control operation request over the default control pipe of this [UsbDevice](#).*
- virtual std::ostream & [describe](#) (std::ostream &os, unsigned int flags=DefaultProperties) const  
*Insert a description of this [UsbDevice](#) into os.*
- bool [isA](#) (const [InterfaceClass](#) &interfaceClass) const  
*The method returns true, if this [UsbDevice](#) is a device supporting the libusb controlled USB device interface class described by interfaceClass.*
- template<class ForwardIterator>  
bool [isA](#) (ForwardIterator beginInterfaceClass, ForwardIterator endInterfaceClass) const  
*The method template returns true, if this [UsbDevice](#) is a device supporting one of the the libusb controlled USB device interface classes whose description is contained in the half open interval [\*beginInterfaceClass, \*endInterfaceClass).*

### Static Public Member Functions

- static size\_t [enumerate](#) (std::vector< [Descriptor](#) > &descriptors, const [InterfaceClass](#) &interfaceClass, bool append=false, bool quick=false)  
*Enumerate all [Device::Descriptor](#) objects describing libusb controlled USB devices supporting the libusb controlled USB device interface class described by interfaceClass.*
- template<class ForwardIterator>  
static size\_t [enumerate](#) (std::vector< [Descriptor](#) > &descriptors, ForwardIterator beginInterfaceClass, ForwardIterator endInterfaceClass, bool append=false)  
*Enumerate all [Device::Descriptor](#) objects describing libusb controlled USB devices supporting one of the the libusb controlled USB device interface classes whose description is contained in the half open interval [\*beginInterfaceClass, \*endInterfaceClass).*
- static size\_t [enumerateAll](#) (std::vector< [Descriptor](#) > &descriptors)  
*Enumerate all [Device::Descriptor](#) objects describing libusb controlled USB devices.*

### Classes

- struct [InterfaceClass](#)  
*Type describing a libusb controlled USB device interface class.*

## 11.5.2 Member Enumeration Documentation

### 11.5.2.1 enum stm::UsbDevice::DescribeFlags

Describe flags (bitwise orable).

The enumerators of [UsbDevice::DescribeFlags](#) augment the [Device::DescribeFlags](#) further specifying the extent produced by [describe\(\)](#).

#### Enumerator:

*DeviceReleaseNumber* If set, the device release number property is included, else not.

*DeviceBusNumber* If set, the device bus number property is included, else not.

*DeviceManufacturer* If set, the device manufacturer property is included, else not.

*DeviceProduct* If set, the device product property is included, else not.

*DeviceSerialNumber* If set, the device serial number property is included, else not.

*DeviceConfigurations* If set, the device configurations property is included, else not.

*DeviceInterfaces* If set and property names are configured, the device interfaces property is included, else not.

*DeviceAltSettings* If set and property names are configured, the device alternate settings property is included, else not.

*DeviceEndpoints* If set and property names are configured, the device endpoints property is included, else not.

*DeviceChildren* If set, the device children property is included, else not.

Reimplemented from [stm::Device](#).

Definition at line 208 of file [usbdevice.hpp](#).

## 11.5.3 Constructor & Destructor Documentation

### 11.5.3.1 stm::UsbDevice::UsbDevice (int *defaultTimeout* = Forever, const Descriptor & *descr* = Descriptor ()) [inline]

Constructor of a [UsbDevice](#) object representing a libusb controlled USB device described by *descr*.

#### Parameters:

← *defaultTimeout* Timeout in milliseconds used by default for all operations of this [UsbDevice](#).

← *descr* A valid [Device::Descriptor](#) for the libusb controlled USB device to be represented by the [UsbDevice](#) object to be constructed or an invalid [Device::Descriptor](#). A valid [Device::Descriptor](#) is typically yielded by one of the static methods [enumerate\(\)](#) or [enumerateAll\(\)](#).

#### Effects:

Constructs a [UsbDevice](#) object with the [Device::Descriptor](#) *descr* defined for it. If *descr* is valid, the constructed [UsbDevice](#) is ready to be opened.

#### See also:

[descr\(\)](#), [open\(\)](#), [enumerate\(\)](#), [enumerateAll\(\)](#).

Definition at line 313 of file [xusbdevice.hpp](#).

### 11.5.3.2 `stm::UsbDevice::~~UsbDevice ()` [inline, virtual]

Destructor.

#### Effects:

If this `UsbDevice` is open, it is closed first.

#### See also:

`isOpen()`, `close()`.

Definition at line 322 of file `xusbdevice.hpp`.

References `close()`, and `isOpen()`.

## 11.5.4 Member Function Documentation

### 11.5.4.1 `bool stm::UsbDevice::canRead () const` [inline, virtual]

Return the read capability of this `UsbDevice`.

#### Returns:

`true`.

#### Note:

If this virtual method is not reimplemented by a derived class, this means that the device can be successfully opened in open mode `Device::ReadAccess`.

It is not necessary that this `UsbDevice` is open.

#### See also:

`canWrite()`, `canControl()`, `canSeek()`, `open()`.

Reimplemented from `stm::Device`.

Definition at line 331 of file `xusbdevice.hpp`.

### 11.5.4.2 `bool stm::UsbDevice::canWrite () const` [inline, virtual]

Return the write capability of this `UsbDevice`.

#### Returns:

`true`.

#### Note:

If this virtual method is not reimplemented by a derived class, this means that the device can be successfully opened in open mode `Device::WriteAccess`.

It is not necessary that this `UsbDevice` is open.

#### See also:

`canRead()`, `canControl()`, `canSeek()`, `open()`.

Reimplemented from `stm::Device`.

Definition at line 337 of file `xusbdevice.hpp`.

#### 11.5.4.3 `bool stm::UsbDevice::canControl () const` [`inline`, `virtual`]

Return the control capability of this `UsbDevice`.

##### Returns:

`true`.

##### Note:

If this virtual method is not reimplemented by a derived class, this means that the device does support the method `control()`.

It is not necessary that this `UsbDevice` is open.

##### See also:

`canRead()`, `canWrite()`, `canSeek()`, `control()`.

Reimplemented from `stm::Device`.

Definition at line 343 of file `xusbdevice.hpp`.

#### 11.5.4.4 `void stm::UsbDevice::setError (int error, const std::string & msg = std::string ()) const` [`virtual`]

Set the error state and error string of this `UsbDevice` according to *error* and *msg*.

##### Parameters:

← *error* Error state as one of the enumerators of `Device::ErrorState` optionally ored with one ore more of the enumerators of `Device::ErrorFlags`.

← *msg* Error string.

##### Effects:

If the error state part of *error* is one of the enumerators of `Device::ErrorState`, the error state of this `UsbDevice` is set to that state and its error string to *msg*, else to `Device::UnknownError`. If the error flag `Device::SystemError` is set in *error*, the error string is augmented by a system error description, if available.

##### Note:

Despite of being `const`, the method can change the error state and error string.

It is not necessary that this `UsbDevice` is open.

##### See also:

`error()`, `clearError()`, `errorString()`, `augmentErrorString()`.

Reimplemented from `stm::Device`.

Definition at line 634 of file `usbdevice.cpp`.

References `stm::Device::NoError`, `stm::Device::setError()`, and `stm::Device::SystemError`.

Referenced by `control()`, `open()`, `read()`, `readPipe()`, `write()`, and `writePipe()`.

**11.5.4.5 void stm::UsbDevice::setDescr (const Descriptor & descr) [inline, virtual]**

Set the [Device::Descriptor](#) of this [UsbDevice](#) to *descr*.

**Parameters:**

- ← *descr* A valid [Device::Descriptor](#) of the libusb controlled USB device to be represented by this [UsbDevice](#) or an invalid [Device::Descriptor](#). A valid descriptor is typically yielded by one of the static methods [enumerate\(\)](#) or [enumerateAll\(\)](#).

**Effects:**

If this [UsbDevice](#) is open, it is closed. Then *descr* is defined for it. If *descr* is valid, this [UsbDevice](#) is ready to be opened, else it does not represent a libusb controlled USB device.

**See also:**

[descr\(\)](#), [isOpen\(\)](#), [close\(\)](#), [open\(\)](#), [enumerate\(\)](#), [enumerateAll\(\)](#).

Implements [stm::Device](#).

Definition at line 349 of file [xusbdevice.hpp](#).

References [stm::Device::descr\(\)](#), [stm::UsbPipe::Invalid](#), [stm::local::xUsbDevice::pipe\\_](#), [stm::local::xUsbDevice::pipeType\\_](#), and [stm::Device::setDescr\(\)](#).

**11.5.4.6 UsbPipe stm::UsbDevice::pipe () const [inline]**

Return a copy of the [UsbPipe](#) object configured for this [UsbDevice](#).

**Returns:**

an invalid [UsbPipe](#) object, if this [UsbDevice](#) does not represent a libusb controlled USB device.  
a copy of the [UsbPipe](#) object configured for the libusb controlled USB device represented by this [UsbDevice](#).

**Note:**

It is not necessary that this [UsbDevice](#) is open.

**See also:**

[pipeType\(\)](#), [setPipe\(\)](#), [isOpen\(\)](#).

Definition at line 362 of file [xusbdevice.hpp](#).

References [stm::local::xUsbDevice::pipe\\_](#).

**11.5.4.7 UsbPipe::Type stm::UsbDevice::pipeType () const [inline]**

Return the type of the [UsbPipe](#) object configured for this [UsbDevice](#) as an enumerator of the enumeration [UsbPipe::Type](#).

**Returns:**

[UsbPipe::Invalid](#), if this [UsbDevice](#) does not represent a libusb controlled USB device, or if its configured pipe is invalid.  
[UsbPipe::Bulk](#) or [UsbPipe::Interrupt](#), if this [UsbDevice](#) represents a libusb controlled USB device with a pipe configured for bulk or interrupt data transfer.

**Note:**

It is not necessary that this [UsbDevice](#) is open.

**See also:**

[pipe\(\)](#), [setPipe\(\)](#), [isOpen\(\)](#).

Definition at line 368 of file `xusbdevice.hpp`.

References `stm::local::xUsbDevice::pipeType_`.

**11.5.4.8 UsbPipe::Type stm::UsbDevice::setPipe (UsbPipe *pipe*)**

Set the UsbPipe object configured for this [UsbDevice](#) to a copy of *pipe* and return its type.

**Parameters:**

← *pipe* A UsbPipe object a copy of which is to be configured for the libusb controlled USB device represented by this [UsbDevice](#).

**Returns:**

The type [UsbPipe::Bulk](#) or [UsbPipe::Interrupt](#), if a copy of *pipe* can be configured for this [UsbDevice](#), else [UsbPipe::Invalid](#), in which case the UsbPipe object configured for this [UsbDevice](#) stays unchanged.

**Note:**

It is not necessary that this [UsbDevice](#) is open.

**See also:**

[pipe\(\)](#), [pipeType\(\)](#), [isOpen\(\)](#).

Definition at line 674 of file `usbdevice.cpp`.

References `stm::UsbPipe::altsetting`, `stm::UsbPipe::Bulk`, `stm::UsbPipe::configuration`, `stm::Device::descr()`, `stm::UsbPipe::endpoint`, `stm::UsbPipe::interface`, `stm::UsbPipe::Interrupt`, `stm::UsbPipe::Invalid`, `stm::local::xUsbDevice::pipe_`, and `stm::local::xUsbDevice::pipeType_`.

Referenced by `readPipe()`, and `writePipe()`.

**11.5.4.9 bool stm::UsbDevice::isOpen () const [inline, virtual]**

Determine, if this [UsbDevice](#) is open.

**Returns:**

`true`, if this [UsbDevice](#) is open, that is if its open mode is not the [Device::OpenMode](#) enumerator [Device::NoAccess](#).

`false`, if this [UsbDevice](#) is not open, that is if its open mode is the [Device::OpenMode](#) enumerator [Device::NoAccess](#).

**See also:**

[open\(\)](#), [close\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 374 of file `xusbdevice.hpp`.

References `stm::local::xUsbDevice::handle_`, and `stm::Device::isOpen()`.

Referenced by `~UsbDevice()`.

#### 11.5.4.10 `bool stm::UsbDevice::open (unsigned int openMode)` [virtual]

Open this [UsbDevice](#) in *openMode*.

##### Parameters:

← *openMode* Open mode to be set.

##### Effects:

The method sets the error state of this [UsbDevice](#) to the [Device::ErrorState](#) enumerator [Device::OpenError](#), if *openMode* is the [Device::OpenMode](#) enumerator [Device::NoAccess](#), if `isOpen()` does not return false or if the libusb controlled USB device represented by this [UsbDevice](#) cannot be opened conforming to *openMode*. Else the method sets the open mode of this [UsbDevice](#) to *openMode*.

##### Returns:

`true`, if this [UsbDevice](#) could be opened in *openMode*.  
`false`, if this [UsbDevice](#) could not be opened in *openMode*. Then the error state of this [UsbDevice](#) is set to [Device::OpenError](#).

##### See also:

[isOpen\(\)](#), [close\(\)](#), [error \(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 729 of file `usbdevice.cpp`.

References `stm::Device::close()`, `stm::local::xUsbDevice::handle_`, `stm::local::UsbHandle::Invalid`, `stm::Device::open()`, `stm::Device::OpenError`, `stm::Device::property()`, and `setError()`.

#### 11.5.4.11 `bool stm::UsbDevice::close ()` [virtual]

Close this [UsbDevice](#).

##### Effects:

The method sets the error state of this [UsbDevice](#) to the [Device::ErrorState](#) enumerator [Device::CloseError](#), if `isOpen()` returns false or if the libusb controlled USB device represented by this [UsbDevice](#) cannot be closed successfully. Else the method sets the open mode of this [UsbDevice](#) to the [Device::OpenMode](#) enumerator [Device::NoAccess](#).

##### Returns:

`true`, if this [UsbDevice](#) could be closed successfully.  
`false`, if this [UsbDevice](#) could not be closed successfully. Then the error state of this [UsbDevice](#) is set to [Device::CloseError](#).

##### See also:

[isOpen\(\)](#), [open\(\)](#), [error \(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 765 of file usbdevice.cpp.

References [stm::Device::close\(\)](#), and [stm::local::xUsbDevice::handle\\_](#).

Referenced by [~UsbDevice\(\)](#).

**11.5.4.12** `int64_t stm::UsbDevice::read (void * data, int64_t maxLen, int timeout = DefaultTimeout, unsigned int flags = NoFlags) [virtual]`

Read *maxLen* bytes from the currently configured USB pipe of this [UsbDevice](#) into the *data* buffer.

**Parameters:**

- *data* Buffer for the data to be read.
- ← *maxLen* Maximal number of bytes to be read.
- ← *timeout* Operation timeout in milliseconds. If the value is [Device::Forever](#), no timeout occurs. The default value [Device::DefaultTimeout](#) means, that the default timeout of this [UsbDevice](#) is used.
- ← *flags* If the flag bit [Device::AcceptTimeout](#) is set, a timeout is no error.

**Effects:**

The method sets the error state of this [UsbDevice](#) to the [Device::ErrorState](#) enumerator [Device::ReadError](#), if the result of [openMode\(\)](#) does not contain the [Device::OpenMode](#) enumerator [Device::ReadAccess](#), or if *maxLen* is negative or *data* is the NULL pointer unless *maxLen* is also, 0 or if the read operation described below fails. During the read operation maximal *maxLen* bytes from the libusb controlled USB device represented by this [UsbDevice](#) are read through the USB pipe currently configured and are stored in the buffer pointed to by *data*.

**Returns:**

The number of bytes actually read, if the read operation was successful.  
 -1, if the read operation was not successful. Then the error state of this [UsbDevice](#) is set to [Device::ReadError](#).

**Note:**

Be aware that in multithreaded applications configuring the USB pipe to be used and the reading from the pipe must occur atomically. To ensure this, better use [readPipe\(\)](#) in those situations.

**See also:**

[write\(\)](#), [control\(\)](#), [openMode\(\)](#), [error \(\)](#), [defaultTimeout\(\)](#), [pipe\(\)](#), [setPipe\(\)](#), [readPipe\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 779 of file usbdevice.cpp.

References [stm::Device::AcceptTimeout](#), [stm::UsbPipe::altsetting](#), [stm::UsbPipe::Bulk](#), [stm::UsbPipe::configuration](#), [stm::Device::defaultTimeout\(\)](#), [stm::Device::DefaultTimeout](#), [stm::UsbPipe::endpoint](#), [ETIMEDOUT](#), [stm::local::xUsbDevice::handle\(\)](#), [stm::local::xUsbDevice::handle\\_](#), [stm::UsbPipe::interface](#), [stm::UsbPipe::Invalid](#), [stm::local::xUsbDevice::pipe\\_](#), [stm::local::xUsbDevice::pipeType\\_](#), [stm::Device::property\(\)](#), [stm::Device::read\(\)](#), [stm::Device::ReadError](#), [setError\(\)](#), and [stm::Device::SystemError](#).

Referenced by [readPipe\(\)](#).

**11.5.4.13** `int64_t stm::UsbDevice::readPipe (UsbPipe pipe, void * data, int64_t maxLen, int timeout = DefaultTimeout, unsigned int flags = NoFlags) [virtual]`

Atomically set the UsbPipe object configured for this [UsbDevice](#) to a copy of *pipe* and read *maxLen* bytes from that pipe into the *data* buffer.

**Parameters:**

- ← *pipe* A UsbPipe object a copy of which is to be configured for the libusb controlled USB device represented by this [UsbDevice](#).
- *data* Buffer for the data to be read.
- ← *maxLen* Maximal number of bytes to be read.
- ← *timeout* Operation timeout in milliseconds. If the value is [Device::Forever](#), no timeout occurs. The default value [Device::DefaultTimeout](#) means, that the default timeout of this [UsbDevice](#) is used.
- ← *flags* If the flag bit [Device::AcceptTimeout](#) is set, a timeout is no error.

**Effects:**

The method sets the error state of this [UsbDevice](#) to the [Device::ErrorState](#) enumerator [Device::ReadError](#), if it cannot set the *pipe* successfully, if the result of [openMode\(\)](#) does not contain the [Device::OpenMode](#) enumerator [Device::ReadAccess](#), or if *maxLen* is negative or *data* is the NULL pointer unless *maxLen* is also, 0 or if the read operation described below fails. During the read operation maximal *maxLen* bytes from the libusb controlled USB device represented by this [UsbDevice](#) are read through *pipe* and are stored in the buffer pointed to by *data*.

**Returns:**

The number of bytes actually read, if the read operation was successful.  
 -1, if the read operation was not successful. Then the error state of this [UsbDevice](#) is set to [Device::ReadError](#).

**Note:**

The *timeout* only applies to the read operation not to the setting of the *pipe*.

**See also:**

[write\(\)](#), [control\(\)](#), [openMode\(\)](#), [error \(\)](#), [defaultTimeout\(\)](#), [pipe\(\)](#), [setPipe\(\)](#), [read\(\)](#).

Definition at line 889 of file `usbdevice.cpp`.

References [stm::local::xUsbDevice::handle\\_](#), [stm::UsbPipe::Invalid](#), [stm::Device::property\(\)](#), [read\(\)](#), [stm::Device::ReadError](#), [setError\(\)](#), and [setPipe\(\)](#).

**11.5.4.14** `int64_t stm::UsbDevice::write (const void * data, int64_t len, int timeout = DefaultTimeout, unsigned int flags = NoFlags) [virtual]`

Write *len* bytes from the *data* buffer to the currently configured USB pipe of this [UsbDevice](#).

**Parameters:**

- ← *data* Buffer containing the data to be written.
- ← *len* Number of bytes to be written.
- ← *timeout* Operation timeout in milliseconds. If the value is [Device::Forever](#), no timeout occurs. The default value [Device::DefaultTimeout](#) means, that the default timeout of this [UsbDevice](#) is used.

← *flags* If the flag bit [Device::AcceptTimeout](#) is set, a timeout is no error.

#### Effects:

The method sets the error state of this [UsbDevice](#) to the [Device::ErrorState](#) enumerator [Device::WriteError](#), if the result of [openMode\(\)](#) does not contain the [Device::OpenMode](#) enumerator [Device::WriteAccess](#), or if *len* is negative or *data* is the NULL pointer unless *len* is also 0, or if the write operation described below fails. During the write operation *len* bytes from the buffer pointed to by *data* are written through the USB pipe currently configured to the libusb controlled USB device represented by this [UsbDevice](#).

#### Returns:

*len*, if the write operation was successful.

-1, if the write operation was not successful. Then the error state of this [UsbDevice](#) is set to [Device::WriteError](#).

#### Note:

Be aware that in multithreaded applications configuring the USB pipe to be used and the writing to the pipe must occur atomically. To ensure this, better use [writePipe\(\)](#) in those situations.

#### See also:

[read\(\)](#), [control\(\)](#), [openMode\(\)](#), [error\(\)](#), [defaultTimeout\(\)](#), [pipe\(\)](#), [setPipe\(\)](#), [writePipe\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 921 of file [usbdevice.cpp](#).

References [stm::Device::AcceptTimeout](#), [stm::UsbPipe::altsetting](#), [stm::UsbPipe::Bulk](#), [stm::UsbPipe::configuration](#), [stm::Device::defaultTimeout\(\)](#), [stm::Device::DefaultTimeout](#), [stm::UsbPipe::endpoint](#), [ETIMEDOUT](#), [stm::local::xUsbDevice::handle\(\)](#), [stm::local::xUsbDevice::handle\\_](#), [stm::UsbPipe::interface](#), [stm::UsbPipe::Invalid](#), [stm::local::xUsbDevice::pipe\\_](#), [stm::local::xUsbDevice::pipeType\\_](#), [stm::Device::property\(\)](#), [setError\(\)](#), [stm::Device::SystemError](#), [stm::Device::write\(\)](#), and [stm::Device::WriteError](#).

Referenced by [writePipe\(\)](#).

#### 11.5.4.15 int64\_t stm::UsbDevice::writePipe (UsbPipe pipe, const void \* data, int64\_t len, int timeout = DefaultTimeout, unsigned int flags = NoFlags) [virtual]

Atomically set the [UsbPipe](#) object configured for this [UsbDevice](#) to a copy of *pipe* and write *len* bytes from the *data* buffer to that pipe.

#### Parameters:

← *pipe* A [UsbPipe](#) object a copy of which is to be configured for the libusb controlled USB device represented by this [UsbDevice](#).

← *data* Buffer containing the data to be written.

← *len* Number of bytes to be written.

← *timeout* Operation timeout in milliseconds. If the value is [Device::Forever](#), no timeout occurs. The default value [Device::DefaultTimeout](#) means, that the default timeout of this [UsbDevice](#) is used.

← *flags* If the flag bit [Device::AcceptTimeout](#) is set, a timeout is no error.

**Effects:**

The method sets the error state of this `UsbDevice` to the `Device::ErrorState` enumerator `Device::WriteError`, if it cannot set the *pipe* successfully, if the result of `openMode()` does not contain the `Device::OpenMode` enumerator `Device::WriteAccess`, or if *len* is negative or *data* is the NULL pointer unless *len* is also 0, or if the write operation described below fails. During the write operation *len* bytes from the buffer pointed to by *data* are written through *pipe* to the libusb controlled USB device represented by this `UsbDevice`.

**Returns:**

*len*, if the write operation was successful.  
 -1, if the write operation was not successful. Then the error state of this `UsbDevice` is set to `Device::WriteError`.

**Note:**

The *timeout* only applies to the write operation not to the setting of the *pipe*.

**See also:**

`read()`, `control()`, `openMode()`, `error ()`, `defaultTimeout()`, `pipe()`, `setPipe()`, `write()`.

Definition at line 1037 of file `usbdevice.cpp`.

References `stm::local::xUsbDevice::handle_`, `stm::UsbPipe::Invalid`, `stm::Device::property()`, `setError()`, `setPipe()`, `write()`, and `stm::Device::WriteError`.

**11.5.4.16** `int64_t stm::UsbDevice::control (unsigned int request, const void * ctrl, int64_t ctrlLen, void * data, int64_t dataLen, int timeout = DefaultTimeout, unsigned int flags = NoFlags) const [virtual]`

Perform the control operation *request* over the default control pipe of this `UsbDevice`.

**Parameters:**

- ← *request* Specifies the particular control operation to be performed.
- ← *ctrl* Pointer to a `UsbCtrl` object specifying the request type (in which the transfer direction is encoded), the value and the index of the request.
- ← *ctrlLen* `sizeof(UsbCtrl)`.
- ↔ *data* Buffer for the input or output data.
- ← *dataLen* Byte length of the data buffer.
- ← *timeout* Operation timeout in milliseconds. If the value is `Device::Forever`, no timeout occurs. The default value `Device::DefaultTimeout` means, that the default timeout of this `UsbDevice` is used.
- ← *flags* If the flag bit `Device::AcceptTimeout` is set, a timeout is no error.

**Effects:**

The method performs the control operation characterized by *request* for the libusb controlled USB device represented by this `UsbDevice` using its default control pipe. The parameter *ctrl* shall be the address of a `UsbCtrl` object specifying the request type encoding the transfer direction, the value and the index of the request. The parameter *ctrlLen* shall be `sizeof(UsbCtrl)`. If any data transfer is required, *data* shall not be NULL and point to a buffer of size *dataLen*.

**Returns:**

The number of bytes transferred to or from *data*, if the operation was successful. This is 0 in the case of an accepted timeout.

-1, if the operation was not successful. Then the error state of this [UsbDevice](#) is set to the [Device::ErrorState](#) enumerator [Device::ControlError](#).

**See also:**

[write\(\)](#), [read\(\)](#), [openMode\(\)](#), [error \(\)](#), [defaultTimeout\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 1069 of file `usbdevice.cpp`.

References [stm::Device::AcceptTimeout](#), [stm::Device::control\(\)](#), [stm::Device::ControlError](#), [stm::Device::defaultTimeout\(\)](#), [stm::Device::DefaultTimeout](#), [ETIMEDOUT](#), [stm::local::xUsbDevice::handle\(\)](#), [stm::local::xUsbDevice::handle\\_](#), [stm::Device::property\(\)](#), [setError\(\)](#), and [stm::Device::SystemError](#).

#### 11.5.4.17 `std::ostream & stm::UsbDevice::describe (std::ostream & os, unsigned int flags = DefaultProperties) const` [`inline, virtual`]

Insert a description of this [UsbDevice](#) into *os*.

**Parameters:**

← *os* The output stream to insert the description.

← *flags* Description flags.

**Effects:**

The method inserts a verbal description of this [UsbDevice](#) into the output stream *os*. Format and extent of the description is controlled by the *flags* parameter according to the bitwise ored enumerators of [Device::DescribeFlags](#) and [UsbDevice::DescribeFlags](#).

**Returns:**

The output stream *os*.

**Note:**

This [UsbDevice](#) need not be open.

Reimplemented from [stm::Device](#).

Definition at line 380 of file `xusbdevice.hpp`.

References [stm::Device::describe\(\)](#), and [stm::local::xUsbDevice::printDevice\(\)](#).

#### 11.5.4.18 `bool stm::UsbDevice::isA (const InterfaceClass & interfaceClass) const`

The method returns true, if this [UsbDevice](#) is a device supporting the libusb controlled USB device interface class described by *interfaceClass*.

That means it returns true, if the [Device::Descriptor](#) of this [UsbDevice](#) describes a libusb controlled USB device supporting the libusb controlled USB device interface class described by *interfaceClass*, else false.

Definition at line 1165 of file `usbdevice.cpp`.

References `stm::Device::descr()`, `stm::UsbDevice::InterfaceClass::productId`, and `stm::UsbDevice::InterfaceClass::vendorId`.

Referenced by `enumerate()`, and `isA()`.

#### 11.5.4.19 `template<class ForwardIterator> bool stm::UsbDevice::isA (ForwardIterator beginInterfaceClass, ForwardIterator endInterfaceClass) const` [inline]

The method template returns true, if this `UsbDevice` is a device supporting one of the the libusb controlled USB device interface classes whose description is contained in the half open interval [`*beginInterfaceClass`, `*endInterfaceClass`).

That means it returns true, if the `Device::Descriptor` of this `UsbDevice` describes a libusb controlled USB device supporting one of the libusb controlled USB device interface classes described by that interval, else false.

Definition at line 393 of file `xusbdevice.hpp`.

References `isA()`.

#### 11.5.4.20 `size_t stm::UsbDevice::enumerate (std::vector< Descriptor > & descriptors, const InterfaceClass & interfaceClass, bool append = false, bool quick = false)` [static]

Enumerate all `Device::Descriptor` objects describing libusb controlled USB devices supporting the libusb controlled USB device interface class described by `interfaceClass`.

The static method clears the vector `descriptors`, scans the system for all libusb controlled USB devices supporting the libusb controlled USB device interface class described by `interfaceClass`, stores the `Device::Descriptor` objects describing those devices in the vector `descriptors` and returns the size of that vector. If `quick` is true, the system is not scanned for new hardware.

Definition at line 1178 of file `usbdevice.cpp`.

References `stm::Device::Forever`, `stm::local::xUsbDevice::initialize()`, and `isA()`.

Referenced by `enumerate()`.

#### 11.5.4.21 `template<class ForwardIterator> size_t stm::UsbDevice::enumerate (std::vector< Descriptor > & descriptors, ForwardIterator beginInterfaceClass, ForwardIterator endInterfaceClass, bool append = false)` [inline, static]

Enumerate all `Device::Descriptor` objects describing libusb controlled USB devices supporting one of the the libusb controlled USB device interface classes whose description is contained in the half open interval [`*beginInterfaceClass`, `*endInterfaceClass`).

The static method template clears the vector `descriptors`, scans the system for all libusb controlled USB devices supporting one of the libusb controlled USB device interface classes described by that interval, stores the `Device::Descriptor` objects describing those devices in the vector `descriptors` and returns the size of that vector.

Definition at line 418 of file `xusbdevice.hpp`.

References `enumerate()`.

#### 11.5.4.22 `size_t stm::UsbDevice::enumerateAll (std::vector< Descriptor > & descriptors)` [static]

Enumerate all `Device::Descriptor` objects describing libusb controlled USB devices.

The static method template clears the vector *descriptors*, scans the system for all libusb controlled USB devices, stores the [Device::Descriptor](#) objects describing those devices in the vector *descriptors* and returns the size of that vector.

Definition at line 1206 of file usbdevice.cpp.

References [stm::local::xUsbDevice::initialize\(\)](#).

The documentation for this class was generated from the following files:

- [usbdevice.hpp](#)
- [xusbdevice.hpp](#)
- [usbdevice.cpp](#)

## 11.6 stm::UsbDevice::InterfaceClass Struct Reference

```
#include <usbdevice.hpp>
```

### 11.6.1 Detailed Description

Type describing a libusb controlled USB device interface class.

Such a libusb controlled USB device interface class is characterized by its vendor and product IDs.

Definition at line 750 of file usbdevice.hpp.

#### Public Member Functions

- [InterfaceClass](#) (unsigned short [vendorId](#)=0, unsigned short [productId](#)=0)  
*Constructor yielding a [UsbDevice::InterfaceClass](#) object describing the libusb controlled USB device interface class characterized by its vendor and product IDs.*

#### Public Attributes

- unsigned short [vendorId](#)
- unsigned short [productId](#)

### 11.6.2 Constructor & Destructor Documentation

#### 11.6.2.1 [stm::UsbDevice::InterfaceClass::InterfaceClass](#) (unsigned short *vendorId* = 0, unsigned short *productId* = 0) [inline]

Constructor yielding a [UsbDevice::InterfaceClass](#) object describing the libusb controlled USB device interface class characterized by its vendor and product IDs.

Definition at line 299 of file xusbdevice.hpp.

### 11.6.3 Member Data Documentation

#### 11.6.3.1 unsigned short [stm::UsbDevice::InterfaceClass::vendorId](#)

Definition at line 757 of file usbdevice.hpp.

Referenced by [stm::UsbDevice::isA\(\)](#).

### 11.6.3.2 unsigned short stm::UsbDevice::InterfaceClass::productId

Definition at line 758 of file usbdevice.hpp.

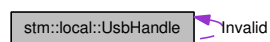
Referenced by stm::UsbDevice::isA().

The documentation for this struct was generated from the following files:

- [usbdevice.hpp](#)
- [xusbdevice.hpp](#)

## 11.7 stm::local::UsbHandle Class Reference

Collaboration diagram for stm::local::UsbHandle:



### 11.7.1 Detailed Description

Definition at line 93 of file usbdevice.cpp.

#### Public Member Functions

- [UsbHandle](#) (const [Device::Descriptor](#) &descr)
- [~UsbHandle](#) ()

#### Static Public Attributes

- static [UsbHandle](#) \* [Invalid](#) = reinterpret\_cast<[UsbHandle](#) \*> (-1L)

#### Private Attributes

- struct usb\_dev\_handle \* [dev\\_](#)
- [StmMutex](#) [mutex\\_](#)

#### Friends

- class [xUsbDevice](#)
- class [Lock](#)

#### Classes

- class [Lock](#)

### 11.7.2 Constructor & Destructor Documentation

#### 11.7.2.1 stm::local::UsbHandle::UsbHandle (const Device::Descriptor & descr) [inline]

Definition at line 107 of file usbdevice.cpp.

References [dev\\_](#), and [mutex\\_](#).

### 11.7.2.2 stm::local::UsbHandle::~~UsbHandle () [inline]

Definition at line 127 of file usbdevice.cpp.

References `dev_`, and `mutex_`.

## 11.7.3 Friends And Related Function Documentation

### 11.7.3.1 friend class xUsbDevice [friend]

Definition at line 95 of file usbdevice.cpp.

### 11.7.3.2 friend class Lock [friend]

Definition at line 96 of file usbdevice.cpp.

## 11.7.4 Member Data Documentation

### 11.7.4.1 UsbHandle \* stm::local::UsbHandle::Invalid = reinterpret\_cast<UsbHandle \*> (-1L) [static]

Definition at line 133 of file usbdevice.cpp.

Referenced by `stm::UsbDevice::open()`.

### 11.7.4.2 struct usb\_dev\_handle\* stm::local::UsbHandle::dev\_ [read, private]

Definition at line 136 of file usbdevice.cpp.

Referenced by `UsbHandle()`, and `~UsbHandle()`.

### 11.7.4.3 StmMutex stm::local::UsbHandle::mutex\_ [private]

Definition at line 137 of file usbdevice.cpp.

Referenced by `UsbHandle()`, and `~UsbHandle()`.

The documentation for this class was generated from the following file:

- [usbdevice.cpp](#)

## 11.8 stm::local::UsbHandle::Lock Class Reference

### 11.8.1 Detailed Description

Definition at line 99 of file usbdevice.cpp.

#### Public Member Functions

- [Lock \(UsbHandle \\*handle\)](#)

## 11.8.2 Constructor & Destructor Documentation

### 11.8.2.1 stm::local::UsbHandle::Lock::Lock (UsbHandle \* *handle*) [inline]

Definition at line 102 of file usbdevice.cpp.

The documentation for this class was generated from the following file:

- [usbdevice.cpp](#)

## 11.9 stm::UsbPipe Struct Reference

```
#include <usbdevice.hpp>
```

### 11.9.1 Detailed Description

Type specifying the configuration, the interface, the alternate setting, and the endpoint of the USB pipe to be used for a USB bulk or interrupt transfer.

Definition at line 123 of file usbdevice.hpp.

#### Public Types

- enum [Type](#) {  
    [Invalid](#) = -1,  
    [Bulk](#),  
    [Interrupt](#) }

#### Public Member Functions

- [UsbPipe](#) ()
- [UsbPipe](#) (unsigned char [configuration](#), unsigned char [interface](#), unsigned char [endpoint](#))
- [UsbPipe](#) (unsigned char [configuration](#), unsigned char [interface](#), unsigned char [altsetting](#), unsigned char [endpoint](#))
- int [type](#) () const
- [operator bool](#) () const

#### Public Attributes

- unsigned char [configuration](#)
- unsigned char [interface](#)
- unsigned char [altsetting](#)
- unsigned char [endpoint](#)

## 11.9.2 Member Enumeration Documentation

### 11.9.2.1 enum stm::UsbPipe::Type

Enumerator:

*Invalid*

*Bulk*

*Interrupt*

Definition at line 125 of file usbdevice.hpp.

### 11.9.3 Constructor & Destructor Documentation

#### 11.9.3.1 stm::UsbPipe::UsbPipe ()

Definition at line 622 of file usbdevice.cpp.

#### 11.9.3.2 stm::UsbPipe::UsbPipe (unsigned char *configuration*, unsigned char *interface*, unsigned char *endpoint*) [inline]

Definition at line 246 of file xusbdevice.hpp.

#### 11.9.3.3 stm::UsbPipe::UsbPipe (unsigned char *configuration*, unsigned char *interface*, unsigned char *altsetting*, unsigned char *endpoint*) [inline]

Definition at line 259 of file xusbdevice.hpp.

### 11.9.4 Member Function Documentation

#### 11.9.4.1 int stm::UsbPipe::type () const

#### 11.9.4.2 stm::UsbPipe::operator bool () const [inline]

Definition at line 272 of file xusbdevice.hpp.

### 11.9.5 Member Data Documentation

#### 11.9.5.1 unsigned char stm::UsbPipe::configuration

Definition at line 152 of file usbdevice.hpp.

Referenced by stm::UsbDevice::read(), stm::UsbDevice::setPipe(), and stm::UsbDevice::write().

#### 11.9.5.2 unsigned char stm::UsbPipe::interface

Definition at line 153 of file usbdevice.hpp.

Referenced by stm::UsbDevice::read(), stm::UsbDevice::setPipe(), and stm::UsbDevice::write().

#### 11.9.5.3 unsigned char stm::UsbPipe::altsetting

Definition at line 154 of file usbdevice.hpp.

Referenced by stm::UsbDevice::read(), stm::UsbDevice::setPipe(), and stm::UsbDevice::write().

### 11.9.5.4 unsigned char stm::UsbPipe::endpoint

Definition at line 155 of file usbdevice.hpp.

Referenced by `stm::UsbDevice::read()`, `stm::UsbDevice::setPipe()`, and `stm::UsbDevice::write()`.

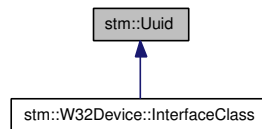
The documentation for this struct was generated from the following files:

- [usbdevice.hpp](#)
- [xusbdevice.hpp](#)
- [usbdevice.cpp](#)

## 11.10 stm::Uuid Struct Reference

```
#include <device.hpp>
```

Inheritance diagram for `stm::Uuid`:



### 11.10.1 Detailed Description

Universal unique identifier storing its fields in little endian format.

Definition at line 307 of file device.hpp.

#### Public Member Functions

- `Uuid ()`  
*Construct the null `Uuid` object with all octet bytes cleared.*
- `Uuid (const unsigned char array[Size])`  
*Construct the `Uuid` object defined by the octet bytes array.*
- `Uuid (unsigned long l, unsigned short w1, unsigned short w2, unsigned char b1, unsigned char b2, unsigned char b3, unsigned char b4, unsigned char b5, unsigned char b6, unsigned char b7, unsigned char b8)`  
*Construct an `Uuid` object from the components `l`, `w1`, `w2`, `b1`, `b2`, `b3`, `b4`, `b5`, `b6`, `b7` and `b8`.*
- `Uuid (const Uuid &other)`  
*Copy constructor.*
- `Uuid & operator= (const Uuid &other)`  
*Assignment operator.*
- `std::string string () const`  
*Return the string representation of this `Uuid`.*

- bool `isNull ()` const  
*Returns true, if this `Uuid` is null.*
- bool `operator== (const Uuid &other)` const  
*Equality comparison operator.*
- bool `operator!= (const Uuid &other)` const  
*Unequality comparison operator.*
- bool `operator< (const Uuid &other)` const  
*Less than comparison operator.*
- bool `operator> (const Uuid &other)` const  
*Greater than comparison operator.*
- bool `operator<= (const Uuid &other)` const  
*Less or equal comparison operator.*
- bool `operator>= (const Uuid &other)` const  
*Greater or equal comparison operator.*

### Public Attributes

- unsigned char `octet [Size]`  
*Structured byte array.*

### Static Public Attributes

- static const size\_t `Size = 16`  
*Number of bytes of a `Uuid` object.*

## 11.10.2 Constructor & Destructor Documentation

### 11.10.2.1 `stm::Uuid::Uuid ()` [inline]

Construct the null `Uuid` object with all octet bytes cleared.

Definition at line 228 of file `xdevice.hpp`.

References `octet`, and `Size`.

Referenced by `isNull()`.

### 11.10.2.2 `stm::Uuid::Uuid (const unsigned char array[Size])` [inline, explicit]

Construct the `Uuid` object defined by the octet bytes `array`.

Definition at line 234 of file `xdevice.hpp`.

References `octet`.

**11.10.2.3** `stm::Uuid::Uuid (unsigned long l, unsigned short w1, unsigned short w2, unsigned char b1, unsigned char b2, unsigned char b3, unsigned char b4, unsigned char b5, unsigned char b6, unsigned char b7, unsigned char b8)` `[inline]`

Construct an [Uuid](#) object from the components *l*, *w1*, *w2*, *b1*, *b2*, *b3*, *b4*, *b5*, *b6*, *b7* and *b8*.

Definition at line 241 of file `xdevice.hpp`.

References `octet`.

**11.10.2.4** `stm::Uuid::Uuid (const Uuid & other)` `[inline]`

Copy constructor.

Definition at line 274 of file `xdevice.hpp`.

References `octet`, and `Size`.

### 11.10.3 Member Function Documentation

**11.10.3.1** `Uuid & stm::Uuid::operator= (const Uuid & other)` `[inline]`

Assignment operator.

Definition at line 280 of file `xdevice.hpp`.

References `octet`, and `Size`.

**11.10.3.2** `std::string stm::Uuid::string () const` `[inline]`

Return the string representation of this [Uuid](#).

Definition at line 287 of file `xdevice.hpp`.

References `octet`.

Referenced by `stm::Device::describe()`.

**11.10.3.3** `bool stm::Uuid::isNull () const` `[inline]`

Returns true, if this [Uuid](#) is null.

That means, if all octet bytes are cleared.

Definition at line 312 of file `xdevice.hpp`.

References `Uuid()`.

**11.10.3.4** `bool stm::Uuid::operator== (const Uuid & other) const` `[inline]`

Equality comparison operator.

Definition at line 318 of file `xdevice.hpp`.

References `octet`, and `Size`.

**11.10.3.5** `bool stm::Uuid::operator!= (const Uuid & other) const` `[inline]`

Unequality comparison operator.

Definition at line 324 of file `xdevice.hpp`.

References `octet`, and `Size`.

#### 11.10.3.6 `bool stm::Uuid::operator< (const Uuid & other) const` [inline]

Less than comparison operator.

Definition at line 330 of file `xdevice.hpp`.

References `octet`, and `Size`.

#### 11.10.3.7 `bool stm::Uuid::operator> (const Uuid & other) const` [inline]

Greater than comparison operator.

Definition at line 336 of file `xdevice.hpp`.

References `octet`, and `Size`.

#### 11.10.3.8 `bool stm::Uuid::operator<= (const Uuid & other) const` [inline]

Less or equal comparison operator.

Definition at line 342 of file `xdevice.hpp`.

References `octet`, and `Size`.

#### 11.10.3.9 `bool stm::Uuid::operator>= (const Uuid & other) const` [inline]

Greater or equal comparison operator.

Definition at line 348 of file `xdevice.hpp`.

References `octet`, and `Size`.

### 11.10.4 Member Data Documentation

#### 11.10.4.1 `const size_t stm::Uuid::Size = 16` [static]

Number of bytes of a `Uuid` object.

Definition at line 310 of file `device.hpp`.

Referenced by `stm::W32Device::InterfaceClass::InterfaceClass()`, `operator!=()`, `operator<()`, `operator<=()`, `operator=()`, `operator==(())`, `operator>()`, `operator>=()`, and `Uuid()`.

#### 11.10.4.2 `unsigned char stm::Uuid::octet[Size]`

Structured byte array.

Its fields are stored in little endian format. In the description below the layout of its hexadecimal bytes `hh` is shown together with the corresponding *octet* array indexes.

```
{ hh . hh . hh . hh-hh . hh-hh . hh-hh . hh-hh-hh-hh-hh-hh-hh }
  3  2  1  0  5  4  7  6  8  9  a  b  c  d  e  f
```

Definition at line 373 of file `device.hpp`.

Referenced by `stm::W32Device::InterfaceClass::InterfaceClass()`, `stm::W32Device::isA()`, `operator!=()`, `operator<()`, `operator<=()`, `operator=()`, `operator==(())`, `operator>()`, `operator>=()`, `string()`, and `Uuid()`.

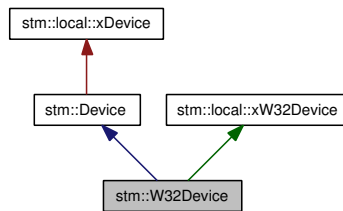
The documentation for this struct was generated from the following files:

- [device.hpp](#)
- [xdevice.hpp](#)

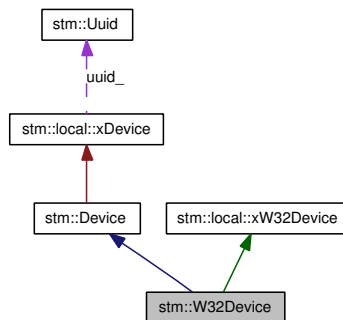
## 11.11 stm::W32Device Class Reference

```
#include <w32device.hpp>
```

Inheritance diagram for stm::W32Device:



Collaboration diagram for stm::W32Device:



### 11.11.1 Detailed Description

Class defining the C++ API for a Windows device inheriting the generic `stm::Device` C++ API.

This class can be instantiated or be used as base class of a special Windows device class which may reimplement the interface of `stm::W32Device` as well as that of the abstract base class `stm::Device`.

Definition at line 128 of file `w32device.hpp`.

#### Public Types

- enum `DescribeFlags` {
  - `DevicePath` = `DriverVersion` << 1,
  - `DeviceInstance` = `DevicePath` << 1,
  - `DeviceIfClass` = `DeviceInstance` << 1 }*Describe flags (bitwise orable).*

### Public Member Functions

- **W32Device** (int defaultTimeout=Forever, const **Descriptor** &descr=**Descriptor**())  
*Constructor of a W32Device object representing a Windows device described by descr.*
- virtual **~W32Device** ()  
*Destructor.*
- virtual bool **canRead** () const  
*Return the read capability of this W32Device.*
- virtual bool **canWrite** () const  
*Return the write capability of this W32Device.*
- virtual bool **canControl** () const  
*Return the control capability of this W32Device.*
- virtual int **error** () const  
*Return the error state of this W32Device.*
- virtual void **setError** (int error, const std::string &msg=std::string()) const  
*Set the error state and error string of this W32Device according to error and msg.*
- virtual void **clearError** () const  
*Clear the error state and error string of this W32Device.*
- virtual bool **isOpen** () const  
*Determine, if this W32Device is open.*
- virtual bool **open** (unsigned int openMode)  
*Open this W32Device in openMode.*
- virtual bool **close** ()  
*Close this W32Device.*
- virtual int64\_t **read** (void \*data, int64\_t maxLen, int timeout=DefaultTimeout, unsigned int flags=NoFlags)  
*Read maxLen bytes from this W32Device into the data buffer.*
- virtual int64\_t **write** (const void \*data, int64\_t len, int timeout=DefaultTimeout, unsigned int flags=NoFlags)  
*Write len bytes from the data buffer to this W32Device.*
- virtual int64\_t **control** (unsigned int request, const void \*inData, int64\_t inLen, void \*outData, int64\_t maxOutLen, int timeout=DefaultTimeout, unsigned int flags=NoFlags) const  
*Perform the control operation request on this W32Device.*
- virtual std::ostream & **describe** (std::ostream &os, unsigned int flags=DefaultProperties) const  
*Insert a description of this W32Device into os.*

- bool [isA](#) (const [InterfaceClass](#) &interfaceClass) const  
*The method returns true, if this [W32Device](#) is a device supporting the Windows device interface class described by interfaceClass.*
- template<class ForwardIterator>  
bool [isA](#) (ForwardIterator beginInterfaceClass, ForwardIterator endInterfaceClass) const  
*The method template returns true, if this [W32Device](#) is a device supporting one of the the Windows device interface classes whose description is contained in the half open interval [\*beginInterfaceClass, \*endInterfaceClass).*

### Static Public Member Functions

- static size\_t [enumerate](#) (std::vector< [Descriptor](#) > &descriptors, const [InterfaceClass](#) &interfaceClass, bool append=false)  
*Enumerate all [Device::Descriptor](#) objects describing Windows devices supporting the Windows device interface class described by interfaceClass.*
- template<class ForwardIterator>  
static size\_t [enumerate](#) (std::vector< [Descriptor](#) > &descriptors, ForwardIterator beginInterfaceClass, ForwardIterator endInterfaceClass, bool append=false)  
*Enumerate all [Device::Descriptor](#) objects describing Windows devices supporting one of the the Windows device interface classes whose description is contained in the half open interval [\*beginInterfaceClass, \*endInterfaceClass).*

### Classes

- struct [InterfaceClass](#)  
*Type describing a Windows device interface class.*

## 11.11.2 Member Enumeration Documentation

### 11.11.2.1 enum stm::W32Device::DescribeFlags

Describe flags (bitwise orable).

The enumerators of [W32Device::DescribeFlags](#) augment the [Device::DescribeFlags](#) further specifying the extent produced by [describe\(\)](#).

#### Enumerator:

***DevicePath*** If set, the device path property is included, else not.

***DeviceInstance*** If set, the device instance property is included, else not.

***DeviceIfClass*** If set, the device interface class property is included, else not.

Reimplemented from [stm::Device](#).

Definition at line 137 of file w32device.hpp.

### 11.11.3 Constructor & Destructor Documentation

#### 11.11.3.1 `stm::W32Device::W32Device (int defaultTimeout = Forever, const Descriptor & descr = Descriptor ()) [inline]`

Constructor of a [W32Device](#) object representing a Windows device described by *descr*.

##### Parameters:

- ← *defaultTimeout* Timeout in milliseconds used by default for all operations of this [W32Device](#).
- ← *descr* A valid [Device::Descriptor](#) of the Windows device to be represented by the [W32Device](#) object to be constructed or an invalid [Device::Descriptor](#). A valid [Device::Descriptor](#) is typically yielded by one of the static methods [enumerate\(\)](#) or [enumerateAll\(\)](#).

##### Effects:

Constructs a [W32Device](#) object with the [Device::Descriptor](#) *descr* defined for it. If *descr* is valid, the constructed [W32Device](#) is ready to be opened.

##### See also:

[descr\(\)](#), [open\(\)](#), [enumerate\(\)](#), [enumerateAll\(\)](#).

Definition at line 195 of file `xw32device.hpp`.

#### 11.11.3.2 `stm::W32Device::~~W32Device () [inline, virtual]`

Destructor.

##### Effects:

If this [W32Device](#) is open, it is closed first.

##### See also:

[isOpen\(\)](#), [close\(\)](#).

Definition at line 204 of file `xw32device.hpp`.

References [close\(\)](#), and [isOpen\(\)](#).

### 11.11.4 Member Function Documentation

#### 11.11.4.1 `bool stm::W32Device::canRead () const [inline, virtual]`

Return the read capability of this [W32Device](#).

##### Returns:

`true`.

##### Note:

If this virtual method is not reimplemented by a derived class, this means that the device can be successfully opened in open mode [Device::ReadAccess](#).

It is not necessary that this [W32Device](#) is open.

**See also:**

[canWrite\(\)](#), [canControl\(\)](#), [canSeek\(\)](#), [open\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 213 of file `xw32device.hpp`.

**11.11.4.2 `bool stm::W32Device::canWrite () const` [inline, virtual]**

Return the write capability of this [W32Device](#).

**Returns:**

`true`.

**Note:**

If this virtual method is not reimplemented by a derived class, this means that the device can be successfully opened in open mode [Device::WriteAccess](#).

It is not necessary that this [W32Device](#) is open.

**See also:**

[canRead\(\)](#), [canControl\(\)](#), [canSeek\(\)](#), [open\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 219 of file `xw32device.hpp`.

**11.11.4.3 `bool stm::W32Device::canControl () const` [inline, virtual]**

Return the control capability of this [W32Device](#).

**Returns:**

`true`.

**Note:**

If this virtual method is not reimplemented by a derived class, this means that the device does support the method [control\(\)](#).

It is not necessary that this [W32Device](#) is open.

**See also:**

[canRead\(\)](#), [canWrite\(\)](#), [canSeek\(\)](#), [control\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 225 of file `xw32device.hpp`.

**11.11.4.4 `int stm::W32Device::error () const` [virtual]**

Return the error state of this [W32Device](#).

**Effects:**

If the error state of this [W32Device](#) is [Device::NoError](#), the Windows last-error code is cleared.

**Returns:**

The error state of this `W32Device` as one of the enumerators of `Device::ErrorState`.

**Note:**

It is not necessary that this `W32Device` is open.

**See also:**

`setError()`, `clearError()`, `errorString()`.

Reimplemented from `stm::Device`.

Definition at line 124 of file `w32device.cpp`.

References `stm::Device::error()`, and `stm::Device::NoError`.

Referenced by `control()`, `read()`, and `write()`.

**11.11.4.5** `void stm::W32Device::setError (int error, const std::string & msg = std::string ()) const` `[virtual]`

Set the error state and error string of this `W32Device` according to *error* and *msg*.

**Parameters:**

- ← *error* Error state as one of the enumerators of `Device::ErrorState` optionally ored with one ore more of the enumerators of `Device::ErrorFlags`.
- ← *msg* Error string.

**Effects:**

If the error state part of *error* is one of the enumerators of `Device::ErrorState`, the error state of this `W32Device` is set to that state and its error string to *msg*, else to `Device::UnknownError`. If the error flag `Device::SystemError` is set in *error*, the error string is augmented by a system error description, if available.

**Note:**

Despite of being `const`, the method can change the error state and error string.  
It is not necessary that this `W32Device` is open.

**See also:**

`error()`, `clearError()`, `errorString()`, `augmentErrorString()`.

Reimplemented from `stm::Device`.

Definition at line 135 of file `w32device.cpp`.

References `stm::Device::NoError`, `stm::Device::setError()`, and `stm::Device::SystemError`.

Referenced by `close()`, `control()`, `open()`, `read()`, and `write()`.

**11.11.4.6** `void stm::W32Device::clearError () const` `[virtual]`

Clear the error state and error string of this `W32Device`.

**Effects:**

The error state and error string of this [W32Device](#) are cleared, that means set to [Device::NoError](#) and the empty string. Moreover, the Windows last-error code is cleared.

**Note:**

Despite of being const, the method can change the error state and error string. It is not necessary that this [W32Device](#) is open.

**See also:**

[error\(\)](#), [setError\(\)](#), [errorString\(\)](#), [augmentErrorString\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 191 of file `w32device.cpp`.

References [stm::Device::clearError\(\)](#).

**11.11.4.7 bool stm::W32Device::isOpen () const** `[inline, virtual]`

Determine, if this [W32Device](#) is open.

**Returns:**

`true`, if this [W32Device](#) is open, that is if its open mode is not the [Device::OpenMode](#) enumerator [Device::NoAccess](#).  
`false`, if this [W32Device](#) is not open, that is if its open mode is the [Device::OpenMode](#) enumerator [Device::NoAccess](#).

**See also:**

[open\(\)](#), [close\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 231 of file `xw32device.hpp`.

References [stm::local::xW32Device::handle\\_](#), and [stm::Device::isOpen\(\)](#).

Referenced by [~W32Device\(\)](#).

**11.11.4.8 bool stm::W32Device::open (unsigned int openMode)** `[virtual]`

Open this [W32Device](#) in *openMode*.

**Parameters:**

← *openMode* Open mode to be set.

**Effects:**

The method sets the error state of this [W32Device](#) to the [Device::ErrorState](#) enumerator [Device::OpenError](#), if *openMode* is the [Device::OpenMode](#) enumerator [Device::NoAccess](#), if [isOpen\(\)](#) does not return false or if the Windows device represented by this [W32Device](#) cannot be opened conforming to *openMode*. Else the method sets the open mode of this [W32Device](#) to *openMode*.

**Returns:**

`true`, if this [W32Device](#) could be opened in *openMode*.  
`false`, if this [W32Device](#) could not be opened in *openMode*. Then the error state of this [W32Device](#) is set to [Device::OpenError](#).

**See also:**

[isOpen\(\)](#), [close\(\)](#), [error \(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 199 of file `w32device.cpp`.

References [stm::Device::close\(\)](#), [stm::Device::descr\(\)](#), [stm::local::xW32Device::handle\\_](#), [stm::Device::open\(\)](#), [stm::Device::OpenError](#), [stm::Device::property\(\)](#), [setError\(\)](#), and [stm::Device::SystemError](#).

**11.11.4.9 bool stm::W32Device::close () [virtual]**

Close this [W32Device](#).

**Effects:**

The method sets the error state of this [W32Device](#) to the [Device::ErrorState](#) enumerator [Device::CloseError](#), if [isOpen\(\)](#) returns false or if the Windows device represented by this [W32Device](#) cannot be closed successfully. Else the method sets the open mode of this [W32Device](#) to the [Device::OpenMode](#) enumerator [Device::NoAccess](#).

**Returns:**

`true`, if this [W32Device](#) could be closed successfully.  
`false`, if this [W32Device](#) could not be closed successfully. Then the error state of this [W32Device](#) is set to [Device::CloseError](#).

**See also:**

[isOpen\(\)](#), [open\(\)](#), [error \(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 254 of file `w32device.cpp`.

References [stm::Device::close\(\)](#), [stm::Device::CloseError](#), [stm::local::xW32Device::handle\\_](#), [stm::Device::property\(\)](#), [setError\(\)](#), and [stm::Device::SystemError](#).

Referenced by [~W32Device\(\)](#).

**11.11.4.10 int64\_t stm::W32Device::read (void \* data, int64\_t maxLen, int timeout = DefaultTimeout, unsigned int flags = NoFlags) [virtual]**

Read *maxLen* bytes from this [W32Device](#) into the *data* buffer.

**Parameters:**

- *data* Buffer for the data to be read.
- ← *maxLen* Maximal number of bytes to be read.
- ← *timeout* Operation timeout in milliseconds. If the value is [Device::Forever](#), no timeout occurs. The default value [Device::DefaultTimeout](#) means, that the default timeout of this [W32Device](#) is used.

← *flags* If the flag bit `Device::AcceptTimeout` is set, a timeout is no error.

**Effects:**

The method sets the error state of this `W32Device` to the `Device::ErrorState` enumerator `Device::ReadError`, if the result of `openMode()` does not contain the `Device::OpenMode` enumerator `Device::ReadAccess`, or if *maxLen* is negative or *data* is the NULL pointer unless *maxLen* is also 0 or if the read operation described below fails. During the read operation maximal *maxLen* bytes from the Windows device represented by this `W32Device` are read and stored in the buffer pointed to by *data*.

**Returns:**

The number of bytes actually read, if the read operation was successful.  
-1, if the read operation was not successful. Then the error state of this `W32Device` is set to `Device::ReadError`.

**See also:**

`write()`, `control()`, `openMode()`, `error ()`, `defaultTimeout()`.

Reimplemented from `stm::Device`.

Definition at line 285 of file `w32device.cpp`.

References `stm::Device::AcceptTimeout`, `stm::Device::defaultTimeout()`, `stm::Device::DefaultTimeout`, `error()`, `stm::Device::Forever`, `stm::local::xW32Device::handle_`, `stm::Device::property()`, `stm::Device::read()`, `stm::Device::ReadError`, `stm::Device::ResourceError`, `setError()`, and `stm::Device::SystemError`.

**11.11.4.11** `int64_t stm::W32Device::write (const void * data, int64_t len, int timeout = DefaultTimeout, unsigned int flags = NoFlags) [virtual]`

Write *len* bytes from the *data* buffer to this `W32Device`.

**Parameters:**

- ← *data* Buffer containing the data to be written.
- ← *len* Number of bytes to be written.
- ← *timeout* Operation timeout in milliseconds. If the value is `Device::Forever`, no timeout occurs. The default value `Device::DefaultTimeout` means, that the default timeout of this `W32Device` is used.
- ← *flags* If the flag bit `Device::AcceptTimeout` is set, a timeout is no error.

**Effects:**

The method sets the error state of this `W32Device` to the `Device::ErrorState` enumerator `Device::WriteError`, if the result of `openMode()` does not contain the `Device::OpenMode` enumerator `Device::WriteAccess`, or if *len* is negative or *data* is the NULL pointer unless *len* is also 0, or if the write operation described below fails. During the write operation *len* bytes from the buffer pointed to by *data* are written to the Windows device represented by this `W32Device`.

**Returns:**

*len*, if the write operation was successful.  
-1, if the write operation was not successful. Then the error state of this `W32Device` is set to `Device::WriteError`.

**See also:**

[read\(\)](#), [control\(\)](#), [openMode\(\)](#), [error \(\)](#), [defaultTimeout\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 414 of file `w32device.cpp`.

References [stm::Device::AcceptTimeout](#), [stm::Device::defaultTimeout\(\)](#), [stm::Device::DefaultTimeout](#), [error\(\)](#), [stm::Device::Forever](#), [stm::local::xW32Device::handle\\_](#), [stm::Device::property\(\)](#), [stm::Device::ResourceError](#), [setError\(\)](#), [stm::Device::SystemError](#), [stm::Device::write\(\)](#), and [stm::Device::WriteError](#).

**11.11.4.12** `int64_t stm::W32Device::control (unsigned int request, const void * inData, int64_t inLen, void * outData, int64_t maxOutLen, int timeout = DefaultTimeout, unsigned int flags = NoFlags) const` [virtual]

Perform the control operation *request* on this [W32Device](#).

**Parameters:**

- ← *request* Specifies the particular control operation to be performed.
- ← *inData* Buffer containing the input data for the control operation.
- ← *inLen* Number of bytes of the input data.
- *outData* Buffer for the output data generated by the control operation.
- ← *maxOutLen* Maximal number of the output data.
- ← *timeout* Operation timeout in milliseconds. If the value is [Device::Forever](#), no timeout occurs. The default value [Device::DefaultTimeout](#) means, that the default timeout of this [W32Device](#) is used.
- ← *flags* If the flag bit [Device::AcceptTimeout](#) is set, a timeout is no error.

**Effects:**

The method performs the control operation characterized by *request* for the Windows device represented by this [W32Device](#). If any input data are required, the parameter *inData* shall point to a buffer of length *inLen* containing them, else *inData* shall be NULL and *inLen* 0. If any data result is expected, *outData* shall not be NULL and point to a buffer of size *maxOutLen*.

**Returns:**

The number of bytes available in *outData*, if the operation was successful. This is 0 in the case of an accepted timeout.

-1, if the operation was not successful. Then the error state of this [W32Device](#) is set to the [Device::ErrorState](#) enumerator [Device::ControlError](#).

**See also:**

[write\(\)](#), [read\(\)](#), [openMode\(\)](#), [error \(\)](#), [defaultTimeout\(\)](#).

Reimplemented from [stm::Device](#).

Definition at line 544 of file `w32device.cpp`.

References [stm::Device::AcceptTimeout](#), [stm::Device::control\(\)](#), [stm::Device::ControlError](#), [stm::Device::defaultTimeout\(\)](#), [stm::Device::DefaultTimeout](#), [error\(\)](#), [stm::Device::Forever](#), [stm::local::xW32Device::handle\\_](#), [stm::Device::property\(\)](#), [stm::Device::ResourceError](#), [setError\(\)](#), and [stm::Device::SystemError](#).

**11.11.4.13** `std::ostream & stm::W32Device::describe (std::ostream & os, unsigned int flags = DefaultProperties) const` `[virtual]`

Insert a description of this `W32Device` into `os`.

**Parameters:**

- ← `os` The output stream to insert the description.
- ← `flags` Description flags.

**Effects:**

The method inserts a verbal description of this `W32Device` into the output stream `os`. Format and extent of the description is controlled by the `flags` parameter according to the bitwise ored enumerators of `Device::DescribeFlags` and `W32Device::DescribeFlags`.

**Returns:**

The output stream `os`.

**Note:**

This `W32Device` need not be open.

Reimplemented from `stm::Device`.

Definition at line 715 of file `w32device.cpp`.

References `stm::Device::AllProperties`, `stm::Device::descr()`, `stm::Device::describe()`, `DeviceIfClass`, `DeviceInstance`, `DevicePath`, `stm::Device::DeviceType`, `stm::Device::DeviceUuid`, `stm::Device::DriverVersion`, `stm::Device::IndentFirst`, `stm::Device::IndentMask`, `stm::Device::NoPropertyNames`, and `stm::Device::VerboseProperties`.

**11.11.4.14** `bool stm::W32Device::isA (const InterfaceClass & interfaceClass) const`

The method returns true, if this `W32Device` is a device supporting the Windows device interface class described by `interfaceClass`.

That means it returns true, if the `Device::Descriptor` of this `W32Device` describes a Windows device supporting the Windows device interface class described by `interfaceClass`, else false.

Definition at line 766 of file `w32device.cpp`.

References `stm::Device::descr()`, and `stm::Uuid::octet`.

Referenced by `isA()`.

**11.11.4.15** `template<class ForwardIterator> bool stm::W32Device::isA (ForwardIterator beginInterfaceClass, ForwardIterator endInterfaceClass) const` `[inline]`

The method template returns true, if this `W32Device` is a device supporting one of the the Windows device interface classes whose description is contained in the half open interval `[*beginInterfaceClass, *endInterfaceClass)`.

That means it returns true, if the `Device::Descriptor` of this `W32Device` describes a Windows device supporting one of the Windows device interface classes described by that interval, else false.

Definition at line 239 of file `xw32device.hpp`.

References `isA()`.

#### 11.11.4.16 `size_t stm::W32Device::enumerate (std::vector< Descriptor > & descriptors, const InterfaceClass & interfaceClass, bool append = false) [static]`

Enumerate all [Device::Descriptor](#) objects describing Windows devices supporting the Windows device interface class described by *interfaceClass*.

The static method clears the vector *descriptors*, scans the system for all Windows devices supporting the Windows device interface class described by *interfaceClass*, stores the [Device::Descriptor](#) objects describing those devices in the vector *descriptors* and returns the size of that vector.

Definition at line 783 of file `w32device.cpp`.

References `DevicePath`, and `stm::local::xW32Device::infoMap_`.

Referenced by `enumerate()`.

#### 11.11.4.17 `template<class ForwardIterator> size_t stm::W32Device::enumerate (std::vector< Descriptor > & descriptors, ForwardIterator beginInterfaceClass, ForwardIterator endInterfaceClass, bool append = false) [inline, static]`

Enumerate all [Device::Descriptor](#) objects describing Windows devices supporting one of the the Windows device interface classes whose description is contained in the half open interval [*\*beginInterfaceClass*, *\*endInterfaceClass*).

The static method template clears the vector *descriptors*, scans the system for all Windows devices supporting one of the Windows device interface classes described by that interval, stores the [Device::Descriptor](#) objects describing those devices in the vector *descriptors* and returns the size of that vector.

Definition at line 264 of file `xw32device.hpp`.

References `enumerate()`.

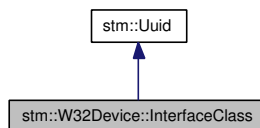
The documentation for this class was generated from the following files:

- [w32device.hpp](#)
- [xw32device.hpp](#)
- [w32device.cpp](#)

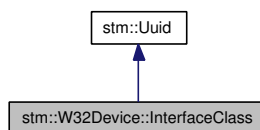
## 11.12 stm::W32Device::InterfaceClass Struct Reference

```
#include <w32device.hpp>
```

Inheritance diagram for `stm::W32Device::InterfaceClass`:



Collaboration diagram for `stm::W32Device::InterfaceClass`:



### 11.12.1 Detailed Description

Type describing a Windows device interface class.

Such a Windows device interface class is characterized by their GUID.

Definition at line 505 of file w32device.hpp.

### Public Member Functions

- [InterfaceClass](#) ()

*Default constructor yielding an invalid [W32Device::InterfaceClass](#) object.*

- [InterfaceClass](#) (const [GUID](#) &interfaceClassGuid)

*Constructor yielding a [W32Device::InterfaceClass](#) object describing the Windows device interface class characterized by their GUID [interfaceClassGuid](#).*

### 11.12.2 Constructor & Destructor Documentation

#### 11.12.2.1 stm::W32Device::InterfaceClass::InterfaceClass ()

Default constructor yielding an invalid [W32Device::InterfaceClass](#) object.

Definition at line 108 of file w32device.cpp.

#### 11.12.2.2 stm::W32Device::InterfaceClass::InterfaceClass (const [GUID](#) & *interfaceClassGuid*)

Constructor yielding a [W32Device::InterfaceClass](#) object describing the Windows device interface class characterized by their GUID *interfaceClassGuid*.

Definition at line 113 of file w32device.cpp.

References [stm::Uuid::octet](#), and [stm::Uuid::Size](#).

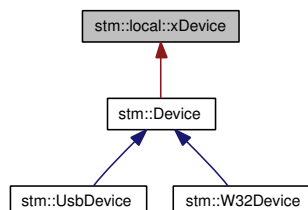
The documentation for this struct was generated from the following files:

- [w32device.hpp](#)
- [w32device.cpp](#)

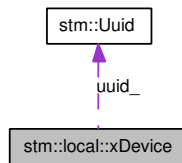
## 11.13 stm::local::xDevice Class Reference

```
#include <xdevice.hpp>
```

Inheritance diagram for [stm::local::xDevice](#):



Collaboration diagram for stm::local::xDevice:



### 11.13.1 Detailed Description

Definition at line 145 of file xdevice.hpp.

#### Protected Types

- typedef [xDevice DeviceImpl](#)

#### Protected Member Functions

- [xDevice](#) (const [Uuid](#) &uuid, const std::string &type, int defaultTimeout, const std::pair< void \*, void(\*)()> &descr)
- [xDevice](#) (const [Uuid](#) &uuid, const std::string &type, const unsigned short version[4], int defaultTimeout, const std::pair< void \*, void(\*)()> &descr)

#### Protected Attributes

- [Uuid](#) [uuid\\_](#)
- std::map< std::string, std::string > [property\\_](#)
- int [defaultTimeout\\_](#)
- std::pair< void \*, void(\*)() [descr\\_](#) )
- unsigned short [version\\_](#) [4]
- int [error\\_](#)
- std::string [errMsg\\_](#)
- unsigned int [openMode\\_](#)

### 11.13.2 Member Typedef Documentation

#### 11.13.2.1 typedef xDevice stm::local::xDevice::DeviceImpl [protected]

Definition at line 148 of file xdevice.hpp.

### 11.13.3 Constructor & Destructor Documentation

#### 11.13.3.1 stm::local::xDevice::xDevice (const [Uuid](#) & *uuid*, const std::string & *type*, int *defaultTimeout*, const std::pair< void \*, void(\*)()> & *descr*) [inline, protected]

Definition at line 151 of file xdevice.hpp.

**11.13.3.2** `stm::local::xDevice::xDevice (const Uuid & uuid, const std::string & type, const unsigned short version[4], int defaultTimeout, const std::pair< void *, void(*)()> & descr)` [inline, protected]

Definition at line 168 of file xdevice.hpp.

#### 11.13.4 Member Data Documentation

**11.13.4.1** `Uuid stm::local::xDevice::uuid_` [protected]

Definition at line 185 of file xdevice.hpp.

Referenced by `stm::Device::setUuid()`, and `stm::Device::uuid()`.

**11.13.4.2** `std::map<std::string, std::string> stm::local::xDevice::property_` [protected]

Definition at line 186 of file xdevice.hpp.

Referenced by `stm::Device::hasProperty()`, `stm::Device::property()`, `stm::Device::setProperty()`, and `stm::Device::unsetProperty()`.

**11.13.4.3** `int stm::local::xDevice::defaultTimeout_` [protected]

Definition at line 187 of file xdevice.hpp.

Referenced by `stm::Device::defaultTimeout()`, and `stm::Device::setDefaultTimeout()`.

**11.13.4.4** `std::pair<void *, void (*) ()> stm::local::xDevice::descr_` [protected]

Definition at line 188 of file xdevice.hpp.

Referenced by `stm::Device::descr()`, and `stm::Device::setDescr()`.

**11.13.4.5** `unsigned short stm::local::xDevice::version_[4]` [protected]

Definition at line 189 of file xdevice.hpp.

Referenced by `stm::Device::setVersion()`, and `stm::Device::version()`.

**11.13.4.6** `int stm::local::xDevice::error_` [mutable, protected]

Definition at line 190 of file xdevice.hpp.

Referenced by `stm::Device::augmentErrorString()`, `stm::Device::clearError()`, `stm::Device::error()`, `stm::Device::errorString()`, and `stm::Device::setError()`.

**11.13.4.7** `std::string stm::local::xDevice::errMsg_` [mutable, protected]

Definition at line 191 of file xdevice.hpp.

Referenced by `stm::Device::augmentErrorString()`, `stm::Device::clearError()`, `stm::Device::error()`, `stm::Device::errorString()`, and `stm::Device::setError()`.

**11.13.4.8** `unsigned int stm::local::xDevice::openMode_` [protected]

Definition at line 192 of file xdevice.hpp.

Referenced by `stm::Device::close()`, `stm::Device::isOpen()`, `stm::Device::open()`, and `stm::Device::openMode()`.

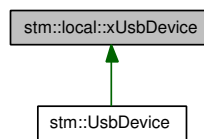
The documentation for this class was generated from the following file:

- [xdevice.hpp](#)

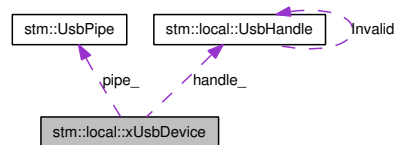
## 11.14 stm::local::xUsbDevice Class Reference

```
#include <xusbdevice.hpp>
```

Inheritance diagram for `stm::local::xUsbDevice`:



Collaboration diagram for `stm::local::xUsbDevice`:



### 11.14.1 Detailed Description

Definition at line 151 of file `xusbdevice.hpp`.

#### Protected Types

- typedef `xUsbDevice` `UsbDeviceImpl`
- typedef struct `usb_device` \* `DeviceDescriptor`
- typedef struct `usb_dev_handle` \* `Handle`
- typedef struct `usb_config_descriptor` \* `ConfigDescriptor`
- typedef struct `usb_interface` \* `Interface`
- typedef struct `usb_interface_descriptor` \* `InterfaceDescriptor`
- typedef struct `usb_endpoint_descriptor` \* `EndpointDescriptor`

#### Protected Member Functions

- `xUsbDevice` ()
- `Handle handle` () const

### Static Protected Member Functions

- static void [printDevice](#) (std::ostream &os, [DeviceDescriptor](#) descr, int level, unsigned int flags)
- static void [printConfig](#) (std::ostream &os, [ConfigDescriptor](#) configDescr, int level, unsigned int flags)
- static void [printInterface](#) (std::ostream &os, [InterfaceDescriptor](#) interfaceDescr, int level, unsigned int flags)
- static void [printEndpoint](#) (std::ostream &os, [EndpointDescriptor](#) endpointDescr, int level, unsigned int flags)
- static void [initialize](#) (bool scanDevices=false)

### Protected Attributes

- [UsbHandle](#) \* handle\_
- [UsbPipe](#) pipe\_
- [UsbPipe::Type](#) pipeType\_

### Static Protected Attributes

- static bool [initialized\\_](#) = false

### Friends

- class [UsbHandle](#)

## 11.14.2 Member Typedef Documentation

### 11.14.2.1 typedef xUsbDevice stm::local::xUsbDevice::UsbDeviceImpl [protected]

Definition at line 156 of file xusbdevice.hpp.

### 11.14.2.2 typedef struct usb\_device\* stm::local::xUsbDevice::DeviceDescriptor [read, protected]

Definition at line 157 of file xusbdevice.hpp.

### 11.14.2.3 typedef struct usb\_dev\_handle\* stm::local::xUsbDevice::Handle [read, protected]

Definition at line 158 of file xusbdevice.hpp.

### 11.14.2.4 typedef struct usb\_config\_descriptor\* stm::local::xUsbDevice::ConfigDescriptor [read, protected]

Definition at line 159 of file xusbdevice.hpp.

### 11.14.2.5 typedef struct usb\_interface\* stm::local::xUsbDevice::Interface [read, protected]

Definition at line 160 of file xusbdevice.hpp.

**11.14.2.6** `typedef struct usb_interface_descriptor* stm::local::xUsbDevice::InterfaceDescriptor`  
[read, protected]

Definition at line 161 of file xusbdevice.hpp.

**11.14.2.7** `typedef struct usb_endpoint_descriptor* stm::local::xUsbDevice::EndpointDescriptor`  
[read, protected]

Definition at line 162 of file xusbdevice.hpp.

### 11.14.3 Constructor & Destructor Documentation

**11.14.3.1** `stm::local::xUsbDevice::xUsbDevice ()` [inline, protected]

Definition at line 164 of file xusbdevice.hpp.

### 11.14.4 Member Function Documentation

**11.14.4.1** `xUsbDevice::Handle stm::local::xUsbDevice::handle () const` [protected]

Definition at line 151 of file usbdevice.cpp.

References `handle_`.

Referenced by `stm::UsbDevice::control()`, `stm::UsbDevice::read()`, and `stm::UsbDevice::write()`.

**11.14.4.2** `void stm::local::xUsbDevice::printDevice (std::ostream & os, DeviceDescriptor descr, int level, unsigned int flags)` [static, protected]

Definition at line 158 of file usbdevice.cpp.

References `stm::UsbDevice::DeviceBusNumber`, `stm::UsbDevice::DeviceChildren`,  
`stm::UsbDevice::DeviceConfigurations`, `stm::UsbDevice::DeviceManufacturer`,  
`stm::UsbDevice::DeviceProduct`, `stm::UsbDevice::DeviceReleaseNumber`,  
`stm::UsbDevice::DeviceSerialNumber`, `stm::Device::DeviceType`, `stm::Device::DeviceUuid`,  
`stm::Device::DriverVersion`, `stm::Device::IndentFirst`, `stm::Device::IndentMask`,  
`stm::Device::NoPropertyNamees`, and `stm::Device::VerboseProperties`.

Referenced by `stm::UsbDevice::describe()`.

**11.14.4.3** `void stm::local::xUsbDevice::printConfig (std::ostream & os, ConfigDescriptor configDescr, int level, unsigned int flags)` [static, protected]

Definition at line 358 of file usbdevice.cpp.

References `stm::UsbDevice::DeviceAltSettings`, `stm::UsbDevice::DeviceInterfaces`, and  
`stm::Device::IndentMask`.

**11.14.4.4** `void stm::local::xUsbDevice::printInterface (std::ostream & os, InterfaceDescriptor interfaceDescr, int level, unsigned int flags)` [static, protected]

Definition at line 460 of file usbdevice.cpp.

References `stm::UsbDevice::DeviceAltSettings`, `stm::UsbDevice::DeviceEndpoints`, and  
`stm::Device::IndentMask`.

**11.14.4.5** void `stm::local::xUsbDevice::printEndpoint` (`std::ostream & os`, `EndpointDescriptor endpointDescr`, `int level`, `unsigned int flags`) [`static`, `protected`]

Definition at line 543 of file `usbdevice.cpp`.

References `stm::UsbDevice::DeviceAltSettings`, and `stm::Device::IndentMask`.

**11.14.4.6** void `stm::local::xUsbDevice::initialize` (`bool scanDevices = false`) [`static`, `protected`]

Definition at line 596 of file `usbdevice.cpp`.

References `initialized_`.

Referenced by `stm::UsbDevice::enumerate()`, and `stm::UsbDevice::enumerateAll()`.

## 11.14.5 Friends And Related Function Documentation

**11.14.5.1** friend class `UsbHandle` [`friend`]

Definition at line 153 of file `xusbdevice.hpp`.

## 11.14.6 Member Data Documentation

**11.14.6.1** `UsbHandle*` `stm::local::xUsbDevice::handle_` [`protected`]

Definition at line 206 of file `xusbdevice.hpp`.

Referenced by `stm::UsbDevice::close()`, `stm::UsbDevice::control()`, `handle()`, `stm::UsbDevice::isOpen()`, `stm::UsbDevice::open()`, `stm::UsbDevice::read()`, `stm::UsbDevice::readPipe()`, `stm::UsbDevice::write()`, and `stm::UsbDevice::writePipe()`.

**11.14.6.2** `UsbPipe` `stm::local::xUsbDevice::pipe_` [`protected`]

Definition at line 207 of file `xusbdevice.hpp`.

Referenced by `stm::UsbDevice::pipe()`, `stm::UsbDevice::read()`, `stm::UsbDevice::setDescr()`, `stm::UsbDevice::setPipe()`, and `stm::UsbDevice::write()`.

**11.14.6.3** `UsbPipe::Type` `stm::local::xUsbDevice::pipeType_` [`protected`]

Definition at line 208 of file `xusbdevice.hpp`.

Referenced by `stm::UsbDevice::pipeType()`, `stm::UsbDevice::read()`, `stm::UsbDevice::setDescr()`, `stm::UsbDevice::setPipe()`, and `stm::UsbDevice::write()`.

**11.14.6.4** `bool` `stm::local::xUsbDevice::initialized_ = false` [`static`, `protected`]

Definition at line 209 of file `xusbdevice.hpp`.

Referenced by `initialize()`.

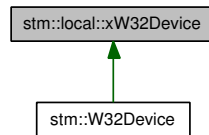
The documentation for this class was generated from the following files:

- [xusbdevice.hpp](#)
- [usbdevice.cpp](#)

## 11.15 stm::local::xW32Device Class Reference

```
#include <xw32device.hpp>
```

Inheritance diagram for stm::local::xW32Device:



### 11.15.1 Detailed Description

Definition at line 128 of file xw32device.hpp.

#### Protected Types

- typedef [xW32Device](#) [W32DeviceImpl](#)
- typedef void \* [Handle](#)
- typedef std::map< std::string, [Ident](#) > [InfoMap](#)
- typedef InfoMap::value\_type [Info](#)
- typedef [Info](#) \* [DeviceDescriptor](#)

#### Protected Member Functions

- [xW32Device](#) ()

#### Protected Attributes

- [Handle](#) [handle\\_](#)

#### Static Protected Attributes

- static [InfoMap](#) [infoMap\\_](#)

#### Classes

- struct [Ident](#)

### 11.15.2 Member Typedef Documentation

#### 11.15.2.1 typedef [xW32Device](#) [stm::local::xW32Device::W32DeviceImpl](#) [protected]

Definition at line 131 of file xw32device.hpp.

#### 11.15.2.2 typedef void\* [stm::local::xW32Device::Handle](#) [protected]

Definition at line 132 of file xw32device.hpp.

**11.15.2.3** `typedef std::map<std::string, Ident> stm::local::xW32Device::InfoMap` [protected]

Definition at line 149 of file xw32device.hpp.

**11.15.2.4** `typedef InfoMap::value_type stm::local::xW32Device::Info` [protected]

Definition at line 150 of file xw32device.hpp.

**11.15.2.5** `typedef Info* stm::local::xW32Device::DeviceDescriptor` [protected]

Definition at line 151 of file xw32device.hpp.

### 11.15.3 Constructor & Destructor Documentation

**11.15.3.1** `stm::local::xW32Device::xW32Device ()` [inline, protected]

Definition at line 153 of file xw32device.hpp.

### 11.15.4 Member Data Documentation

**11.15.4.1** `Handle stm::local::xW32Device::handle_` [protected]

Definition at line 157 of file xw32device.hpp.

Referenced by `stm::W32Device::close()`, `stm::W32Device::control()`, `stm::W32Device::isOpen()`, `stm::W32Device::open()`, `stm::W32Device::read()`, and `stm::W32Device::write()`.

**11.15.4.2** `xW32Device::InfoMap stm::local::xW32Device::infoMap_` [static, protected]

Definition at line 158 of file xw32device.hpp.

Referenced by `stm::W32Device::enumerate()`.

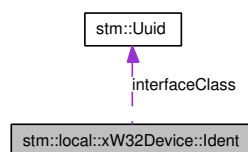
The documentation for this class was generated from the following files:

- [xw32device.hpp](#)
- [w32device.cpp](#)

## 11.16 stm::local::xW32Device::Ident Struct Reference

```
#include <xw32device.hpp>
```

Collaboration diagram for `stm::local::xW32Device::Ident`:



### 11.16.1 Detailed Description

Definition at line 134 of file xw32device.hpp.

#### Public Member Functions

- `Ident` (const `Uuid` &`interfaceClass`=`Uuid`(), const `std::string` &`instanceId`=`std::string`())

#### Public Attributes

- `Uuid` `interfaceClass`
- `std::string` `instanceId`

### 11.16.2 Constructor & Destructor Documentation

**11.16.2.1** `stm::local::xW32Device::Ident::Ident` (const `Uuid` & *interfaceClass* = `Uuid` (), const `std::string` & *instanceId* = `std::string` ()) [inline]

Definition at line 137 of file xw32device.hpp.

### 11.16.3 Member Data Documentation

#### 11.16.3.1 `Uuid` `stm::local::xW32Device::Ident::interfaceClass`

Definition at line 145 of file xw32device.hpp.

#### 11.16.3.2 `std::string` `stm::local::xW32Device::Ident::instanceId`

Definition at line 146 of file xw32device.hpp.

The documentation for this struct was generated from the following file:

- [xw32device.hpp](#)

## 12 SysToMath IO C++ Libraries Implementation File Documentation

### 12.1 config.h File Reference

#### 12.1.1 Detailed Description

Definition of macros for library generation and of pragmas for automatic library choice.

#### Version:

1.06-r11

#### Date:

2006-11-17 17:34:39 (tom)

**Author:**

Tom Michaelis  
 SysToMath  
 Wittelsbacherstr. 7  
 D-80469 Munich

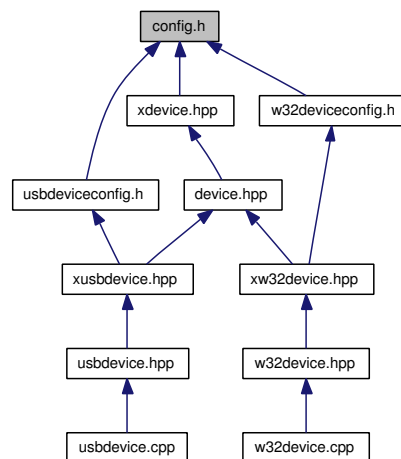
**Contact:**

<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

This header file defines the macro `StmDllSpec` necessary for correct library usage and the pragma comment(lib) responsible for correct library choice.

Definition in file [config.h](#).

This graph shows which files directly or indirectly include this file:

**Defines**

- #define [StmDllSpec](#)

**12.1.2 Define Documentation****12.1.2.1 #define StmDllSpec**

Definition at line 94 of file [config.h](#).

**12.2 device.hpp File Reference****12.2.1 Detailed Description**

Abstract base class [stm::Device](#) forming the ANSI-C++ API for generic devices.

**Version:**

1.03-r41

**Date:**

2007-07-31 09:31:16 (Tom)

**Author:**

Tom Michaelis  
 SysToMath  
 Wittelsbacherstr. 7  
 D-80469 Munich

**Contact:**

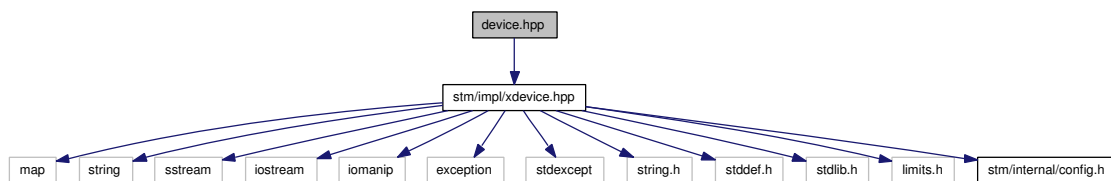
<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

This header file declares the abstract base class `stm::Device`.

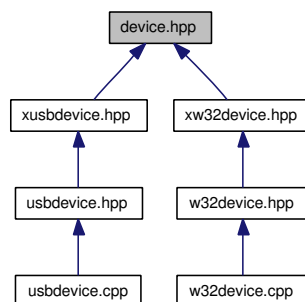
Definition in file [device.hpp](#).

```
#include <stm/impl/xdevice.hpp>
```

Include dependency graph for device.hpp:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace `stm`

**Classes**

- struct `stm::Uuid`  
*Universal unique identifier storing its fields in little endian format.*
- class `stm::Device`

*Abstract base class defining the C++ API for a generic device.*

- struct `stm::Device::Descriptor`

*Objects of type `Device::Descriptor` describe system dependent aspects of a `Device` as a pair of a void pointer and a void function pointer.*

- struct `stm::Device::Version`

*`Device` driver version.*

## Functions

- `std::ostream & stm::operator<< (std::ostream &os, const Device &device)`

*Insert a description of the `Device` device into os.*

## 12.3 usbdevice.cpp File Reference

### 12.3.1 Detailed Description

Base class `stm::UsbDevice` forming the ANSI-C++ API for libusb controlled USB devices.

#### Version:

1.02-r36

#### Date:

2007-07-05 22:45:52 (Tom)

#### Author:

Tom Michaelis  
SysToMath  
Wittelsbacherstr. 7  
D-80469 Munich

#### Contact:

<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

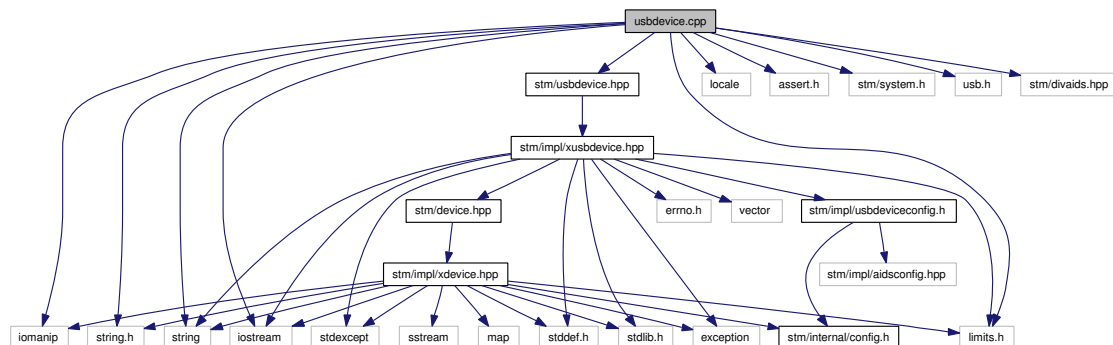
This program file implements the base class `stm::UsbDevice`.

Definition in file `usbdevice.cpp`.

```
#include <string>
#include <locale>
#include <iostream>
#include <iomanip>
#include <assert.h>
#include <limits.h>
```

```
#include <string.h>
#include <stm/system.h>
#include <usb.h>
#include <stm/usbdevice.hpp>
#include <stm/divaids.hpp>
```

Include dependency graph for usbdevice.cpp:



## Namespaces

- namespace [stm](#)
- namespace [stm::local](#)

## Classes

- class [stm::local::UsbHandle](#)
- class [stm::local::UsbHandle::Lock](#)

## Defines

- #define [ETIMEDOUT](#) 116

### 12.3.2 Define Documentation

#### 12.3.2.1 #define ETIMEDOUT 116

Definition at line 81 of file usbdevice.cpp.

Referenced by [stm::UsbDevice::control\(\)](#), [stm::UsbDevice::read\(\)](#), and [stm::UsbDevice::write\(\)](#).

## 12.4 usbdevice.hpp File Reference

### 12.4.1 Detailed Description

Base class [stm::UsbDevice](#) forming the ANSI-C++ API for libusb controlled USB devices.

**Version:**

1.03-r41

**Date:**

2007-07-31 09:31:17 (Tom)

**Author:**

Tom Michaelis  
 SysToMath  
 Wittelsbacherstr. 7  
 D-80469 Munich

**Contact:**

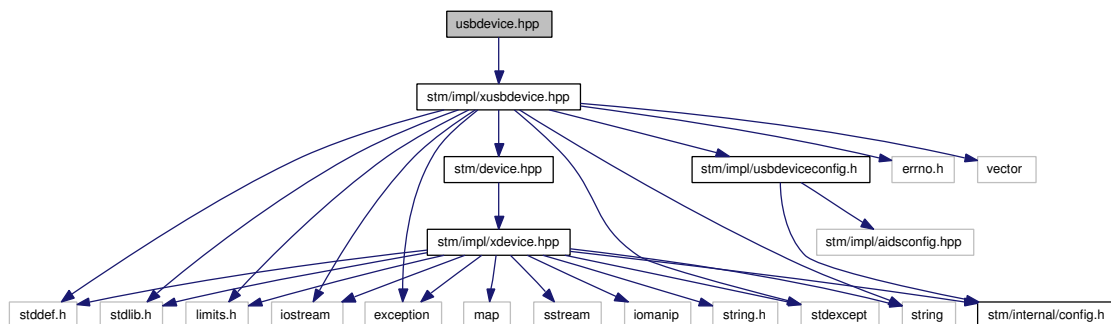
<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

This header file declares the base class `stm::UsbDevice`.

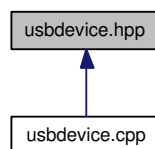
Definition in file `usbdevice.hpp`.

```
#include <stm/impl/xusbdevice.hpp>
```

Include dependency graph for `usbdevice.hpp`:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace `stm`

**Classes**

- struct `stm::UsbPipe`

Type specifying the configuration, the interface, the alternate setting, and the endpoint of the USB pipe to be used for a USB bulk or interrupt transfer.

- struct [stm::UsbCtrl](#)

Type specifying the request type encoding the transfer direction, the value and the index of a USB control request.

- class [stm::UsbDevice](#)

Class defining the C++ API for a libusb controlled USB device inheriting the generic [stm::Device](#) C++ API.

- struct [stm::UsbDevice::InterfaceClass](#)

Type describing a libusb controlled USB device interface class.

## 12.5 usbdeviceconfig.h File Reference

### 12.5.1 Detailed Description

Management of library generation and of automatic library choice for the Artista W32 SDK library stmusbdevice.

**Version:**

1.01-r13

**Date:**

2007-02-18 20:02:14 (Tom)

**Author:**

Tom Michaelis  
SysToMath  
Wittelsbacherstr. 7  
D-80469 Munich

**Contact:**

<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

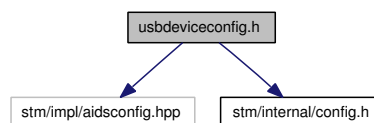
This C++ header file defines the macro `StmDllSpec` necessary for correct library usage and the pragma comment(lib) responsible for correct library choice.

Definition in file [usbdeviceconfig.h](#).

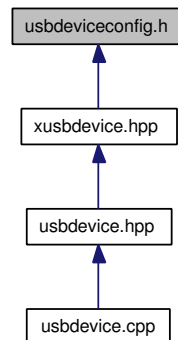
```
#include <stm/impl/aidsconfig.hpp>
```

```
#include <stm/internal/config.h>
```

Include dependency graph for `usbdeviceconfig.h`:



This graph shows which files directly or indirectly include this file:



### Defines

- #define `STM_LIB_NAME` "stmusbdevice"

### 12.5.2 Define Documentation

#### 12.5.2.1 #define STM\_LIB\_NAME "stmusbdevice"

Definition at line 75 of file usbdeviceconfig.h.

## 12.6 w32device.cpp File Reference

### 12.6.1 Detailed Description

Base class `stm::W32Device` forming the ANSI-C++ API for Win32 devices.

#### Version:

1.05-r54

#### Date:

2007-11-05 15:14:09 (Tom)

#### Author:

Tom Michaelis  
SysToMath  
Wittelsbacherstr. 7  
D-80469 Munich

#### Contact:

<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

This program file implements the base class `stm::W32Device`.

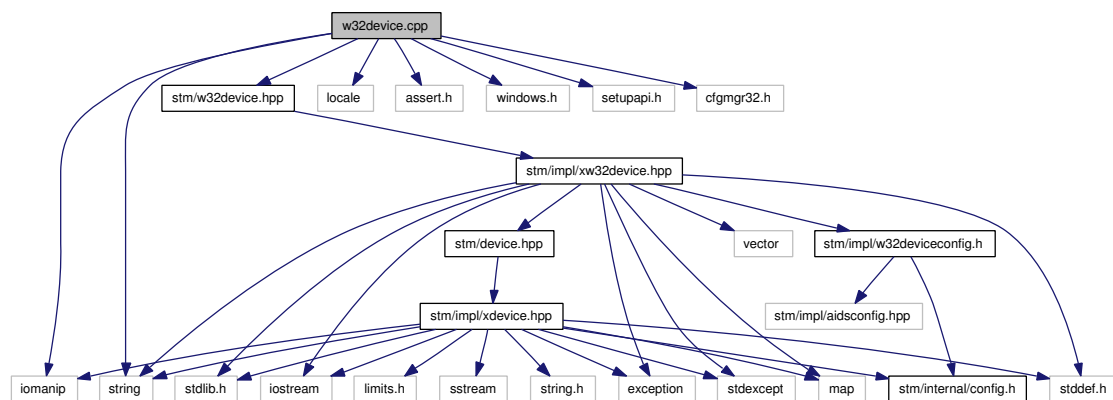
Definition in file `w32device.cpp`.

```

#include <string>
#include <locale>
#include <iomanip>
#include <assert.h>
#include <windows.h>
#include <setupapi.h>
#include <cfgmgr32.h>
#include <stm/w32device.hpp>

```

Include dependency graph for w32device.cpp:



## Namespaces

- namespace [stm](#)
- namespace [stm::local](#)

## 12.7 w32device.hpp File Reference

### 12.7.1 Detailed Description

Base class [stm::W32Device](#) forming the ANSI-C++ API for Win32 devices.

#### Version:

1.01-r28

#### Date:

2007-06-13 01:28:56 (Tom)

#### Author:

Tom Michaelis  
 SysToMath  
 Wittelsbacherstr. 7  
 D-80469 Munich

**Contact:**

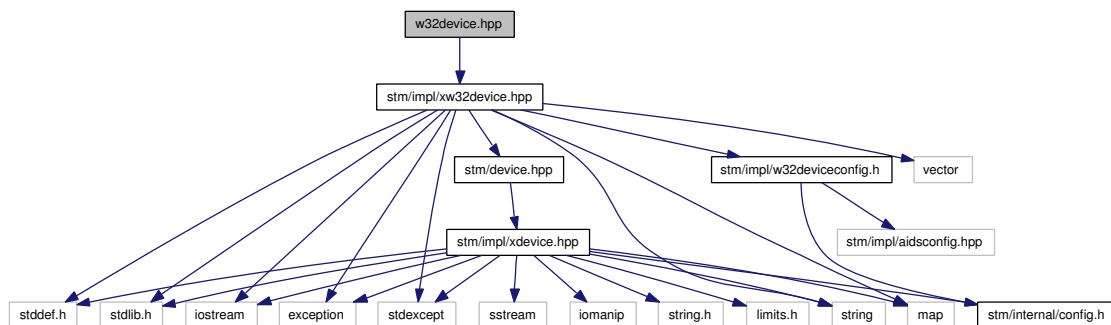
<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

This header file declares the base class [stm::W32Device](#).

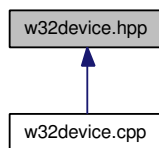
Definition in file [w32device.hpp](#).

```
#include <stm/impl/xw32device.hpp>
```

Include dependency graph for w32device.hpp:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace [stm](#)

**Classes**

- class [stm::W32Device](#)  
*Class defining the C++ API for a Windows device inheriting the generic [stm::Device](#) C++ API.*
- struct [stm::W32Device::InterfaceClass](#)  
*Type describing a Windows device interface class.*

**12.8 w32deviceconfig.h File Reference****12.8.1 Detailed Description**

Management of library generation and of automatic library choice for the Artista W32 SDK library `stmw32device`.

**Version:**

1.01-r13

**Date:**

2007-02-18 20:02:13 (Tom)

**Author:**

Tom Michaelis  
SysToMath  
Wittelsbacherstr. 7  
D-80469 Munich

**Contact:**

<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

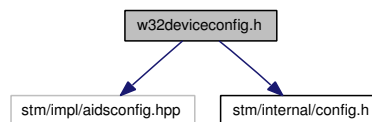
This C++ header file defines the macro `StmDllSpec` necessary for correct library usage and the pragma comment(lib) responsible for correct library choice.

Definition in file [w32deviceconfig.h](#).

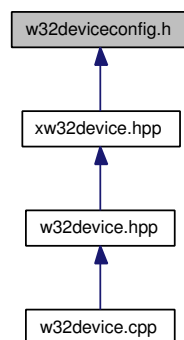
```
#include <stm/impl/aidsconfig.hpp>
```

```
#include <stm/internal/config.h>
```

Include dependency graph for `w32deviceconfig.h`:



This graph shows which files directly or indirectly include this file:

**Defines**

- `#define STM_LIB_NAME "stmw32device"`

## 12.8.2 Define Documentation

### 12.8.2.1 #define STM\_LIB\_NAME "stmw32device"

Definition at line 75 of file w32deviceconfig.h.

## 12.9 xdevice.hpp File Reference

### 12.9.1 Detailed Description

Abstract base class [stm::Device](#) forming the ANSI-C++ API for generic devices.

**Version:**

1.03-r41

**Date:**

2007-07-31 09:31:17 (Tom)

**Author:**

Tom Michaelis  
SysToMath  
Wittelsbacherstr. 7  
D-80469 Munich

**Contact:**

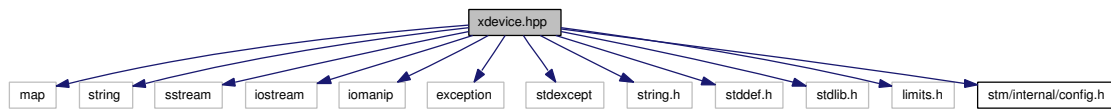
<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

This header file implements the abstract base class [stm::Device](#).

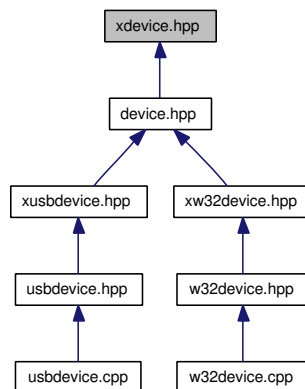
Definition in file [xdevice.hpp](#).

```
#include <map>
#include <string>
#include <sstream>
#include <iostream>
#include <iomanip>
#include <exception>
#include <stdexcept>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include <limits.h>
#include <stm/internal/config.h>
```

Include dependency graph for xdevice.hpp:



This graph shows which files directly or indirectly include this file:



## Namespaces

- namespace [stm](#)
- namespace [stm::local](#)

## Classes

- class [stm::local::xDevice](#)

## Defines

- `#define` [STM\\_XDEVICE\\_HPP](#)

## Functions

- `std::ostream & stm::operator<< (std::ostream &os, const Device &device)`  
*Insert a description of the Device device into os.*

## 12.9.2 Define Documentation

### 12.9.2.1 `#define STM_XDEVICE_HPP`

Definition at line 1192 of file xdevice.hpp.

## 12.10 xusbdevice.hpp File Reference

### 12.10.1 Detailed Description

Base class `stm::UsbDevice` forming the ANSI-C++ API for libusb controlled USB devices.

**Version:**

1.01-r28

**Date:**

2007-06-13 01:28:57 (Tom)

**Author:**

Tom Michaelis  
 SysToMath  
 Wittelsbacherstr. 7  
 D-80469 Munich

**Contact:**

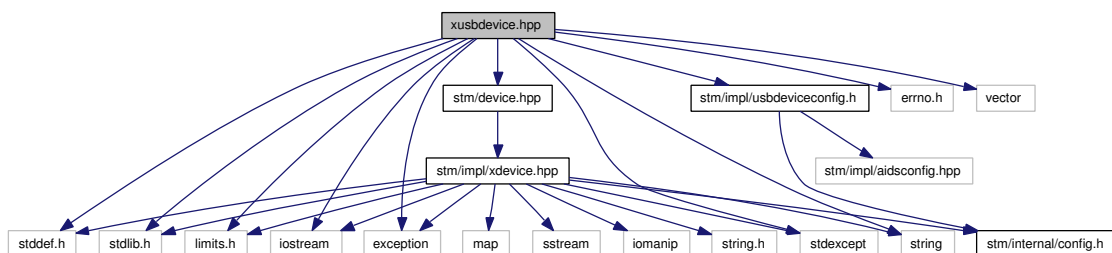
<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

This header file implements the base class `stm::UsbDevice`.

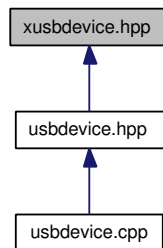
Definition in file `xusbdevice.hpp`.

```
#include <stddef.h>
#include <stdlib.h>
#include <limits.h>
#include <errno.h>
#include <iostream>
#include <exception>
#include <stdexcept>
#include <string>
#include <vector>
#include <stm/device.hpp>
#include <stm/impl/usbdeviceconfig.h>
```

Include dependency graph for `xusbdevice.hpp`:



This graph shows which files directly or indirectly include this file:



### Namespaces

- namespace [stm](#)
- namespace [stm::local](#)

### Classes

- class [stm::local::xUsbDevice](#)

### Defines

- #define [STM\\_XUSBDEVICE\\_HPP](#)

#### 12.10.2 Define Documentation

##### 12.10.2.1 #define STM\_XUSBDEVICE\_HPP

Definition at line 451 of file xusbdevice.hpp.

## 12.11 xw32device.hpp File Reference

### 12.11.1 Detailed Description

Base class [stm::W32Device](#) forming the ANSI-C++ API for Win32 devices.

#### Version:

1.01-r28

#### Date:

2007-06-13 01:28:56 (Tom)

#### Author:

Tom Michaelis  
SysToMath  
Wittelsbacherstr. 7  
D-80469 Munich

**Contact:**

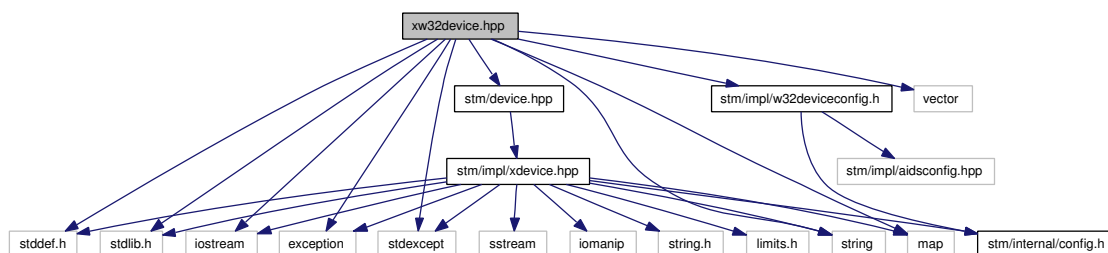
<http://www.SysToMath.com>  
<mailto:Tom.Michaelis@SysToMath.com>

This header file implements the base class `stm::W32Device`.

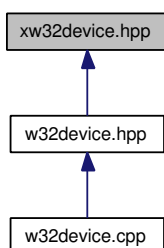
Definition in file `xw32device.hpp`.

```
#include <stddef.h>
#include <stdlib.h>
#include <iostream>
#include <exception>
#include <stdexcept>
#include <string>
#include <vector>
#include <map>
#include <stm/device.hpp>
#include <stm/impl/w32deviceconfig.h>
```

Include dependency graph for `xw32device.hpp`:



This graph shows which files directly or indirectly include this file:

**Namespaces**

- namespace `stm`
- namespace `stm::local`

## Classes

- class [stm::local::xW32Device](#)
- struct [stm::local::xW32Device::Ident](#)

## Defines

- `#define` [STM\\_XW32DEVICE\\_HPP](#)

## Typedefs

- `typedef struct` [\\_GUID](#) [GUID](#)

### 12.11.2 Define Documentation

#### 12.11.2.1 `#define STM_XW32DEVICE_HPP`

Definition at line 295 of file `xw32device.hpp`.

### 12.11.3 Typedef Documentation

#### 12.11.3.1 `typedef struct _GUID` [GUID](#)

Definition at line 101 of file `xw32device.hpp`.

## 13 SysToMath IO C++ Libraries Implementation Page Documentation

### 13.1 Microsoft Visual Studio Tool Family

The Microsoft Visual Studio tool family consists of the tool sets:

- Microsoft Visual Studio .NET 2003 (`vc71`)
- Microsoft Visual Studio 2005 (`vc80`)

#### 13.1.1 Automatic Linking with Microsoft Visual Studio

On Microsoft Visual Studio .NET 2003 (`vc71`) and Microsoft Visual Studio 2005 (`vc80`) the necessary libraries are linked automatically when one of the main library interface header files [stm/w32device.hpp](#) or [stm/usbdevice.hpp](#), unless this mechanism is suppressed by the definition of the preprocessor symbol `STM_NO_LIB`, `STM_W32DEVICE_NO_LIB` or `STM_USBDEVICE_NO_LIB` before that inclusion.

The choice of the libraries depends on the tool set used (`vc71` or `vc80`) and on the system runtime library selected for the executable to be built. The SysToMath IO C++ Libraries package provides for each of its non-header-only library modules four static library configurations (`lib` files) and two dynamic ones (`dll` files). In the following list *module* stands for `w32device` or `usbdevice` and *vcnn* for `vc71` or `vc80`:

- **DebugMt:** Static debug library `libstmmodule-vcnn-mt-gd.lib` together with its debug database file `libstmmodule-vcnn-mt-gd.pdb` chosen when linking against the multithreaded debug DLL runtime (Compiler switch `/MDd`).
- **DebugDIIMt:** Dynamic debug library `stmmodule-vcnn-mt-gd.dll` together with its import library `stmmodule-vcnn-mt-gd.lib` and its debug database file `stmmodule-vcnn-mt-gd.pdb` chosen when linking against the multithreaded debug DLL runtime (Compiler switch `/MDd`) and defining the preprocessor variable `STM_DYN_LINK` before the SysToMath C++ Libraries header file inclusion.
- **DebugMtStaticRt:** Static debug library `libstmmodule-vcnn-mt-sgd.lib` together with its debug database file `libstmmodule-vcnn-mt-sgd.pdb` chosen when linking against the multithreaded static debug runtime (Compiler switch `/MTd`).
- **ReleaseMt:** Static release library `libstmmodule-vcnn-mt.lib` chosen when linking against the multithreaded release DLL runtime (Compiler switch `/MD`).
- **ReleaseDIIMt:** Dynamic release library `stmmodule-vcnn-mt.dll` together with its import library `stmmodule-vcnn-mt.lib` chosen when linking against the multithreaded release DLL runtime (Compiler switch `/MD`) and defining the preprocessor variable `STM_DYN_LINK` before the SysToMath C++ Libraries header file inclusion.
- **ReleaseMtStaticRt:** Static release library `libstmmodule-vcnn-mt-s.lib` chosen when linking against the multithreaded static release runtime (Compiler switch `/MT`).

### 13.1.2 Environment

It is recommended that all static libraries (`lib` files) and their debug database files (`pdb` files) are located in a directory contained in the compiler system library search path. Moreover, to satisfy the application runtime requirements, it is recommended that all dynamic link libraries (`dll` files) and their debug database files (`pdb` files) are located in the application directory or in a directory contained in the system executable search path.

If you used the installation program `LibStmIoSetup.exe` to install the SysToMath IO C++ Libraries with the installation root directory, say `C:\Program Files\SysToMath` and you use Microsoft Visual Studio .NET 2003 (`vc71`) or Microsoft Visual Studio 2005 (`vc80`), then the aforementioned compiler system directory recommendations are satisfied, if you add the following entries in Visual Studio, menu Tools, Options, Projects, VC++ Directories:

- Executable Files: Add `C:\Program Files\SysToMath\bin\w32`
- Library Files: Add `C:\Program Files\SysToMath\lib\w32`
- Include Files: Add `C:\Program Files\SysToMath\include`

## 13.2 GNU Tool Family

The GNU tool family consists of the tool sets:

- GNU `gcc` for POSIX environment (`cygwin`)
- GNU `gcc` for Microsoft environment (`cygming`)

The tool set `cygwin` (chosen by `gcc` or `g++` without option `-mno-cygwin`) produces libraries depending on the dynamic library `cygwin1.dll` and therefore is restricted to free open source software projects, whereas the tool set `cygming` (chosen by `gcc` or `g++` with option `-mno-cygwin`) produces libraries only depending on the Microsoft runtime libraries and thus also allows commercial closed source software projects.

### 13.2.1 Linking with the GNU Tool Family

The libraries for the GNU tool family have to be linked explicitly.

The choice of the libraries depends on the tool set used (`cygwin` or `cygming`) and on the system runtime library selected for the executable to be built. The SysToMath IO C++ Libraries package provides for each of its library modules two static library configurations (`a` files) and two dynamic ones (`dll` files). In the following list `module` stands for `w32device` or `usbdevice` and `cygxxx` for `cygwin` or `cygming`:

- **DebugMt:** Static debug library `libstmmodule-cygxxx-mt-d.a` to be chosen for multithreaded debug enabled statically linked executables (the preprocessor variables `_MT` and `_DEBUG` should be defined).
- **DebugDllMt:** Dynamic debug library `stmmodule-cygxxx-mt-d.dll` together with its import library `libstmmodule-cygxxx-mt-d.dll.a` to be chosen for multithreaded debug enabled dynamically linked executables (the preprocessor variables `_MT`, `_DEBUG` and `STM_DYN_LINK` should be defined).
- **ReleaseMt:** Static release library `libstmmodule-cygxxx-mt.a` to be chosen for multithreaded not debug enabled statically linked executables (the preprocessor variables `_MT` and `NDEBUG` should be defined).
- **ReleaseDllMt:** Dynamic release library `stmmodule-cygxxx-mt.dll` together with its import library `libstmmodule-cygxxx-mt.dll.a` to be chosen for multithreaded not debug enabled dynamically linked executables (the preprocessor variables `_MT`, `NDEBUG` and `STM_DYN_LINK` should be defined).

### 13.2.2 Environment

It is recommended that all static libraries (`a` files) are located in a directory contained in the compiler system library search path. Moreover, to satisfy the application runtime requirements, it is recommended that all dynamic link libraries (`dll` files) are located in the application directory or in a directory contained in the system executable search path.

If you used the installation program `LibStmIoSetup.exe` to install the SysToMath C++ Libraries with the installation root directory, say `C:\Program Files\SysToMath`, and you have added `/cygdrive/c/Program Files/SysToMath/bin/w32` to your `PATH` environment variable, then the aforementioned compiler system directory recommendations are satisfied, if you use the following options in your `gcc` or `g++` command lines:

- Include Files: Use `-I"/cygdrive/c/Program Files/SysToMath/include"`
- Library Files: Use `-L"/cygdrive/c/Program Files/SysToMath/lib/w32"`

## 13.3 SysToMath Device C++ Library (Headers only)

### 13.3.1 Content

The SysToMath Device C++ Library consists of header files only.

### 13.3.2 Usage

To compile an application which uses the [SysToMath Device C++ Library](#), the public main interface header file

- [stm/device.hpp](#)

shall be included and be located in a directory contained in the compiler system include search path. Moreover, the implementation header file

- [stm/impl/xdevice.hpp](#)

shall also be located in a directory contained in the compiler system include search path.

## 13.4 SysToMath UsbDevice C++ Library

### 13.4.1 Content

The SysToMath UsbDevice C++ Library consists of the following object:

- [UsbDevice: Base Class for USB Devices](#)

### 13.4.2 Usage

To compile an application which uses the [SysToMath USB Device C++ Library](#), the public main library interface header file

- [stm/usbdevice.hpp](#)

shall be included and be located in a directory contained in the compiler system include search path. Moreover, the implementation header files

- [stm/impl/usbdeviceconfig.hpp](#)
- [stm/impl/xusbdevice.hpp](#)

shall also be located in a directory contained in the compiler system include search path.

## 13.5 SysToMath W32Device C++ Library

### 13.5.1 Content

The SysToMath W32Device C++ Library consists of the following object:

- [W32Device: Base Class for Win32 Devices](#)

### 13.5.2 Usage

To compile an application which uses the [SysToMath Win32 Device C++ Library](#), the public main library interface header file

- [stm/w32device.hpp](#)

shall be included and be located in a directory contained in the compiler system include search path. Moreover, the implementation header files

- [stm/impl/w32deviceconfig.hpp](#)
- [stm/impl/xw32device.hpp](#)

shall also be located in a directory contained in the compiler system include search path.

## Index

- ~Device
  - stm::Device, 23
- ~UsbDevice
  - stm::UsbDevice, 44
- ~UsbHandle
  - stm::local::UsbHandle, 57
- ~W32Device
  - stm::W32Device, 68
- AcceptTimeout
  - stm::Device, 22
- AllProperties
  - stm::Device, 22
- altsetting
  - stm::UsbPipe, 60
- ArgumentError
  - stm::Device, 21
- augmentErrorString
  - stm::Device, 30
- Bulk
  - stm::UsbPipe, 59
- canControl
  - stm::Device, 28
  - stm::UsbDevice, 45
  - stm::W32Device, 69
- canRead
  - stm::Device, 27
  - stm::UsbDevice, 45
  - stm::W32Device, 68
- canSeek
  - stm::Device, 28
- canWrite
  - stm::Device, 27
  - stm::UsbDevice, 45
  - stm::W32Device, 69
- clearError
  - stm::Device, 30
  - stm::W32Device, 70
- close
  - stm::Device, 32
  - stm::UsbDevice, 49
  - stm::W32Device, 72
- CloseError
  - stm::Device, 21
- config.h, 86
  - StmDllSpec, 87
- ConfigDescriptor
  - stm::local::xUsbDevice, 81
- configuration
  - stm::UsbPipe, 60
- control
  - stm::Device, 33
  - stm::UsbDevice, 53
  - stm::W32Device, 74
- ControlError
  - stm::Device, 21
- DefaultProperties
  - stm::Device, 22
- DefaultTimeout
  - stm::Device, 23
- defaultTimeout
  - stm::Device, 26
- defaultTimeout\_
  - stm::local::xDevice, 79
- descr
  - stm::Device, 26
- descr\_
  - stm::local::xDevice, 79
- describe
  - stm::Device, 34
  - stm::UsbDevice, 54
  - stm::W32Device, 74
- DescribeFlags
  - stm::Device, 22
  - stm::UsbDevice, 44
  - stm::W32Device, 67
- Descriptor
  - stm::Device::Descriptor, 35
- dev\_
  - stm::local::UsbHandle, 58
- Device
  - stm::Device, 23
- Device Implementation, 6
- device.hpp, 87
- Device: Abstract Base Class for Generic Devices, 5
- DeviceAltSettings
  - stm::UsbDevice, 44
- DeviceBusNumber
  - stm::UsbDevice, 44
- DeviceChildren
  - stm::UsbDevice, 44
- DeviceConfigurations
  - stm::UsbDevice, 44
- DeviceDescriptor
  - stm::local::xUsbDevice, 81
  - stm::local::xW32Device, 85
- DeviceEndpoints
  - stm::UsbDevice, 44
- DeviceIfClass

- stm::W32Device, [67](#)
- DeviceImpl
  - stm::local::xDevice, [78](#)
- DeviceInstance
  - stm::W32Device, [67](#)
- DeviceInterfaces
  - stm::UsbDevice, [44](#)
- DeviceManufacturer
  - stm::UsbDevice, [44](#)
- DevicePath
  - stm::W32Device, [67](#)
- DeviceProduct
  - stm::UsbDevice, [44](#)
- DeviceReleaseNumber
  - stm::UsbDevice, [44](#)
- DeviceSerialNumber
  - stm::UsbDevice, [44](#)
- DeviceType
  - stm::Device, [22](#)
- DeviceUuid
  - stm::Device, [22](#)
- DriverVersion
  - stm::Device, [22](#)
- endpoint
  - stm::UsbPipe, [60](#)
- EndpointDescriptor
  - stm::local::xUsbDevice, [82](#)
- enumerate
  - stm::UsbDevice, [55](#)
  - stm::W32Device, [75](#), [76](#)
- enumerateAll
  - stm::UsbDevice, [55](#)
- errMsg\_
  - stm::local::xDevice, [79](#)
- error
  - stm::Device, [29](#)
  - stm::W32Device, [69](#)
- error\_
  - stm::local::xDevice, [79](#)
- ErrorFlagMask
  - stm::Device, [22](#)
- ErrorFlags
  - stm::Device, [21](#)
- ErrorState
  - stm::Device, [21](#)
- errorString
  - stm::Device, [30](#)
- ETIMEDOUT
  - usbdevice.cpp, [90](#)
- Forever
  - stm::Device, [23](#)
- GUID
  - xw32device.hpp, [102](#)
- Handle
  - stm::local::xUsbDevice, [81](#)
  - stm::local::xW32Device, [84](#)
- handle
  - stm::local::xUsbDevice, [82](#)
- handle\_
  - stm::local::xUsbDevice, [83](#)
  - stm::local::xW32Device, [85](#)
- hasProperty
  - stm::Device, [25](#)
- Ident
  - stm::local::xW32Device::Ident, [86](#)
- IndentFirst
  - stm::Device, [22](#)
- IndentMask
  - stm::Device, [22](#)
- index
  - stm::UsbCtrl, [40](#)
- Info
  - stm::local::xW32Device, [85](#)
- InfoMap
  - stm::local::xW32Device, [84](#)
- infoMap\_
  - stm::local::xW32Device, [85](#)
- initialize
  - stm::local::xUsbDevice, [83](#)
- initialized\_
  - stm::local::xUsbDevice, [83](#)
- instanceId
  - stm::local::xW32Device::Ident, [86](#)
- Interface
  - stm::local::xUsbDevice, [81](#)
- interface
  - stm::UsbPipe, [60](#)
- InterfaceClass
  - stm::UsbDevice::InterfaceClass, [56](#)
  - stm::W32Device::InterfaceClass, [77](#)
- interfaceClass
  - stm::local::xW32Device::Ident, [86](#)
- InterfaceDescriptor
  - stm::local::xUsbDevice, [81](#)
- Interrupt
  - stm::UsbPipe, [60](#)
- Invalid
  - stm::local::UsbHandle, [58](#)
  - stm::UsbPipe, [59](#)
- isA
  - stm::UsbDevice, [54](#), [55](#)
  - stm::W32Device, [75](#)
- isNull

- stm::Device::Version, 38
- stm::Uuid, 63
- isOpen
  - stm::Device, 31
  - stm::UsbDevice, 48
  - stm::W32Device, 71
- Lock
  - stm::local::UsbHandle, 58
  - stm::local::UsbHandle::Lock, 59
- Major
  - stm::Device::Version, 37
- Micro
  - stm::Device::Version, 37
- Minor
  - stm::Device::Version, 37
- ModDevice
  - operator<<, 6
- mutex\_
  - stm::local::UsbHandle, 58
- Nano
  - stm::Device::Version, 37
- NoAccess
  - stm::Device, 21
- NoError
  - stm::Device, 21
- NoFlags
  - stm::Device, 22
- NoPropertyNames
  - stm::Device, 22
- octet
  - stm::Uuid, 64
- open
  - stm::Device, 31
  - stm::UsbDevice, 49
  - stm::W32Device, 71
- OpenError
  - stm::Device, 21
- OpenMode
  - stm::Device, 21
- openMode
  - stm::Device, 31
- openMode\_
  - stm::local::xDevice, 79
- operator bool
  - stm::UsbPipe, 60
- operator const void \*
  - stm::Device::Descriptor, 35
- operator!=
  - stm::Device::Version, 38
  - stm::Uuid, 63
- operator<
  - stm::Device::Version, 38
  - stm::Uuid, 64
- operator<<
  - ModDevice, 6
- operator<=
  - stm::Device::Version, 38
  - stm::Uuid, 64
- operator>
  - stm::Device::Version, 38
  - stm::Uuid, 64
- operator>=
  - stm::Device::Version, 38
  - stm::Uuid, 64
- operator=
  - stm::Device, 24
  - stm::Device::Version, 37
  - stm::Uuid, 63
- operator==
  - stm::Device::Version, 38
  - stm::Uuid, 63
- part
  - stm::Device::Version, 39
- Parts
  - stm::Device::Version, 37
- pipe
  - stm::UsbDevice, 47
- pipe\_
  - stm::local::xUsbDevice, 83
- pipeType
  - stm::UsbDevice, 47
- pipeType\_
  - stm::local::xUsbDevice, 83
- pos
  - stm::Device, 33
- printConfig
  - stm::local::xUsbDevice, 82
- printDevice
  - stm::local::xUsbDevice, 82
- printEndpoint
  - stm::local::xUsbDevice, 82
- printInterface
  - stm::local::xUsbDevice, 82
- productId
  - stm::UsbDevice::InterfaceClass, 56
- property
  - stm::Device, 24
- property\_
  - stm::local::xDevice, 79
- proxy
  - stm::Device, 34
- read

- stm::Device, 32
- stm::UsbDevice, 50
- stm::W32Device, 72
- ReadAccess
  - stm::Device, 21
- ReadError
  - stm::Device, 21
- readPipe
  - stm::UsbDevice, 50
- ReadWriteAccess
  - stm::Device, 21
- requestType
  - stm::UsbCtrl, 40
- reset
  - stm::Device, 34
- ResourceError
  - stm::Device, 21
- seek
  - stm::Device, 33
- SeekError
  - stm::Device, 21
- setDefaultTimeout
  - stm::Device, 26
- setDescr
  - stm::Device, 26
  - stm::UsbDevice, 46
- setError
  - stm::Device, 29
  - stm::UsbDevice, 46
  - stm::W32Device, 70
- setPipe
  - stm::UsbDevice, 48
- setProperty
  - stm::Device, 24
- setType
  - stm::Device, 25
- setUuid
  - stm::Device, 24
- setVersion
  - stm::Device, 25
- Size
  - stm::Uuid, 64
- size
  - stm::Device, 33
- stm, 15
- stm::Device, 16
  - ~Device, 23
  - AcceptTimeout, 22
  - AllProperties, 22
  - ArgumentError, 21
  - augmentErrorString, 30
  - canControl, 28
  - canRead, 27
  - canSeek, 28
  - canWrite, 27
  - clearError, 30
  - close, 32
  - CloseError, 21
  - control, 33
  - ControlError, 21
  - DefaultProperties, 22
  - DefaultTimeout, 23
  - defaultTimeout, 26
  - descr, 26
  - describe, 34
  - DescribeFlags, 22
  - Device, 23
  - DeviceType, 22
  - DeviceUuid, 22
  - DriverVersion, 22
  - error, 29
  - ErrorFlagMask, 22
  - ErrorFlags, 21
  - ErrorState, 21
  - errorString, 30
  - Forever, 23
  - hasProperty, 25
  - IndentFirst, 22
  - IndentMask, 22
  - isOpen, 31
  - NoAccess, 21
  - NoError, 21
  - NoFlags, 22
  - NoPropertyNames, 22
  - open, 31
  - OpenError, 21
  - OpenMode, 21
  - openMode, 31
  - operator=, 24
  - pos, 33
  - property, 24
  - proxy, 34
  - read, 32
  - ReadAccess, 21
  - ReadError, 21
  - ReadWriteAccess, 21
  - reset, 34
  - ResourceError, 21
  - seek, 33
  - SeekError, 21
  - setDefaultTimeout, 26
  - setDescr, 26
  - setError, 29
  - setProperty, 24
  - setType, 25
  - setUuid, 24
  - setVersion, 25

- size, 33
- SystemError, 22
- Timeout, 22
- type, 25
- UnknownError, 21
- unsetProperty, 25
- uuid, 24
- VerboseProperties, 22
- version, 25
- write, 32
- WriteAccess, 21
- WriteError, 21
- stm::Device::Descriptor, 35
  - Descriptor, 35
  - operator const void \*, 35
- stm::Device::Version, 35
  - isNull, 38
  - Major, 37
  - Micro, 37
  - Minor, 37
  - Nano, 37
  - operator!=, 38
  - operator<, 38
  - operator<=, 38
  - operator>, 38
  - operator>=, 38
  - operator=, 37
  - operator==, 38
  - part, 39
  - Parts, 37
  - string, 38
  - Version, 37
- stm::local, 15
- stm::local::UsbHandle, 57
  - ~UsbHandle, 57
  - dev\_, 58
  - Invalid, 58
  - Lock, 58
  - mutex\_, 58
  - UsbHandle, 57
  - xUsbDevice, 58
- stm::local::UsbHandle::Lock, 58
  - Lock, 59
- stm::local::xDevice, 77
  - defaultTimeout\_, 79
  - descr\_, 79
  - DeviceImpl, 78
  - errMsg\_, 79
  - error\_, 79
  - openMode\_, 79
  - property\_, 79
  - uuid\_, 79
  - version\_, 79
  - xDevice, 78
- stm::local::xUsbDevice, 80
  - ConfigDescriptor, 81
  - DeviceDescriptor, 81
  - EndpointDescriptor, 82
  - Handle, 81
  - handle, 82
  - handle\_, 83
  - initialize, 83
  - initialized\_, 83
  - Interface, 81
  - InterfaceDescriptor, 81
  - pipe\_, 83
  - pipeType\_, 83
  - printConfig, 82
  - printDevice, 82
  - printEndpoint, 82
  - printInterface, 82
  - UsbDeviceImpl, 81
  - UsbHandle, 83
  - xUsbDevice, 82
- stm::local::xW32Device, 84
  - DeviceDescriptor, 85
  - Handle, 84
  - handle\_, 85
  - Info, 85
  - InfoMap, 84
  - infoMap\_, 85
  - W32DeviceImpl, 84
  - xW32Device, 85
- stm::local::xW32Device::Ident, 85
  - Ident, 86
  - instanceId, 86
  - interfaceClass, 86
- stm::UsbCtrl, 39
  - index, 40
  - requestType, 40
  - UsbCtrl, 39
  - value, 40
- stm::UsbDevice, 40
  - ~UsbDevice, 44
  - canControl, 45
  - canRead, 45
  - canWrite, 45
  - close, 49
  - control, 53
  - describe, 54
  - DescribeFlags, 44
  - DeviceAltSettings, 44
  - DeviceBusNumber, 44
  - DeviceChildren, 44
  - DeviceConfigurations, 44
  - DeviceEndpoints, 44
  - DeviceInterfaces, 44
  - DeviceManufacturer, 44

- DeviceProduct, 44
- DeviceReleaseNumber, 44
- DeviceSerialNumber, 44
- enumerate, 55
- enumerateAll, 55
- isA, 54, 55
- isOpen, 48
- open, 49
- pipe, 47
- pipeType, 47
- read, 50
- readPipe, 50
- setDescr, 46
- setError, 46
- setPipe, 48
- UsbDevice, 44
- write, 51
- writePipe, 52
- stm::UsbDevice::InterfaceClass, 56
  - InterfaceClass, 56
  - productId, 56
  - vendorId, 56
- stm::UsbPipe, 59
  - altsetting, 60
  - Bulk, 59
  - configuration, 60
  - endpoint, 60
  - interface, 60
  - Interrupt, 60
  - Invalid, 59
  - operator bool, 60
  - Type, 59
  - type, 60
  - UsbPipe, 60
- stm::Uuid, 61
  - isNull, 63
  - octet, 64
  - operator!=, 63
  - operator<, 64
  - operator<=, 64
  - operator>, 64
  - operator>=, 64
  - operator=, 63
  - operator==, 63
  - Size, 64
  - string, 63
  - Uuid, 62, 63
- stm::W32Device, 65
  - ~W32Device, 68
  - canControl, 69
  - canRead, 68
  - canWrite, 69
  - clearError, 70
  - close, 72
  - control, 74
  - describe, 74
  - DescribeFlags, 67
  - DeviceIfClass, 67
  - DeviceInstance, 67
  - DevicePath, 67
  - enumerate, 75, 76
  - error, 69
  - isA, 75
  - isOpen, 71
  - open, 71
  - read, 72
  - setError, 70
  - W32Device, 68
  - write, 73
- stm::W32Device::InterfaceClass, 76
  - InterfaceClass, 77
- STM\_LIB\_NAME
  - usbdeviceconfig.h, 93
  - w32deviceconfig.h, 97
- STM\_XDEVICE\_HPP
  - xdevice.hpp, 98
- STM\_XUSBDEVICE\_HPP
  - xusbdevice.hpp, 100
- STM\_XW32DEVICE\_HPP
  - xw32device.hpp, 102
- stmdevice/ Directory Reference, 14
- stmdevice/stm/ Directory Reference, 13
- stmdevice/stm/impl/ Directory Reference, 10
- stmdevice/stm/internal/ Directory Reference, 12
- StmDllSpec
  - config.h, 87
- stmusbdevice/ Directory Reference, 14
- stmusbdevice/stm/ Directory Reference, 13
- stmusbdevice/stm/impl/ Directory Reference, 11
- stmw32device/ Directory Reference, 14
- stmw32device/stm/ Directory Reference, 12
- stmw32device/stm/impl/ Directory Reference, 11
- string
  - stm::Device::Version, 38
  - stm::Uuid, 63
- SystemError
  - stm::Device, 22
- SysToMath Device C++ Library, 5
- SysToMath USB Device C++ Library, 7
- SysToMath Win32 Device C++ Library, 9
- Timeout
  - stm::Device, 22
- Type
  - stm::UsbPipe, 59
- type
  - stm::Device, 25
  - stm::UsbPipe, 60

- UnknownError
  - stm::Device, 21
- unsetProperty
  - stm::Device, 25
- UsbCtrl
  - stm::UsbCtrl, 39
- UsbDevice
  - stm::UsbDevice, 44
- UsbDevice Implementation, 7
- usbdevice.cpp, 89
  - ETIMEDOUT, 90
- usbdevice.hpp, 90
- UsbDevice: Base Class for USB Devices, 8
- usbdeviceconfig.h, 92
  - STM\_LIB\_NAME, 93
- UsbDeviceImpl
  - stm::local::xUsbDevice, 81
- UsbHandle
  - stm::local::UsbHandle, 57
  - stm::local::xUsbDevice, 83
- UsbPipe
  - stm::UsbPipe, 60
- Uuid
  - stm::Uuid, 62, 63
- uuid
  - stm::Device, 24
- uuid\_
  - stm::local::xDevice, 79
  
- value
  - stm::UsbCtrl, 40
- vendorId
  - stm::UsbDevice::InterfaceClass, 56
- VerboseProperties
  - stm::Device, 22
- Version
  - stm::Device::Version, 37
- version
  - stm::Device, 25
- version\_
  - stm::local::xDevice, 79
  
- W32Device
  - stm::W32Device, 68
- W32Device Implementation, 9
- w32device.cpp, 93
- w32device.hpp, 94
- W32Device: Base Class for Win32 Devices, 10
- w32deviceconfig.h, 95
  - STM\_LIB\_NAME, 97
- W32DeviceImpl
  - stm::local::xW32Device, 84
- write
  - stm::Device, 32
  - stm::UsbDevice, 51
  - stm::W32Device, 73
- WriteAccess
  - stm::Device, 21
- WriteError
  - stm::Device, 21
- writePipe
  - stm::UsbDevice, 52
  
- xDevice
  - stm::local::xDevice, 78
- xdevice.hpp, 97
  - STM\_XDEVICE\_HPP, 98
- xUsbDevice
  - stm::local::UsbHandle, 58
  - stm::local::xUsbDevice, 82
- xusbdevice.hpp, 99
  - STM\_XUSBDEVICE\_HPP, 100
- xW32Device
  - stm::local::xW32Device, 85
- xw32device.hpp, 100
  - GUID, 102
  - STM\_XW32DEVICE\_HPP, 102