Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

THIS SPEC IS OBSOLETE

Spec No: 002-05489

Spec Title: AN205489-F2MC-16FX/FR/FM3 USB FAMILY 16/32-BIT USB MICROCONTROLLER USB MCU SERIES FUJITSU USB LIB

Replaced By: NONE

AN205489

F²MC-16FX/FR/FM3 USB Family 16/32-Bit USB Microcontroller USB MCU Series Fujitsu USB Lib

This Application Note describes about FUJITSU USB LIB for F²MC-16FX/FR/FM3 USB FAMILY.

This FujitsuUsbLib is programmed for Microsoft Windows .NET applications. It has backends for the serial port and is only allowed to use with Fujitsu hardware.

Contents

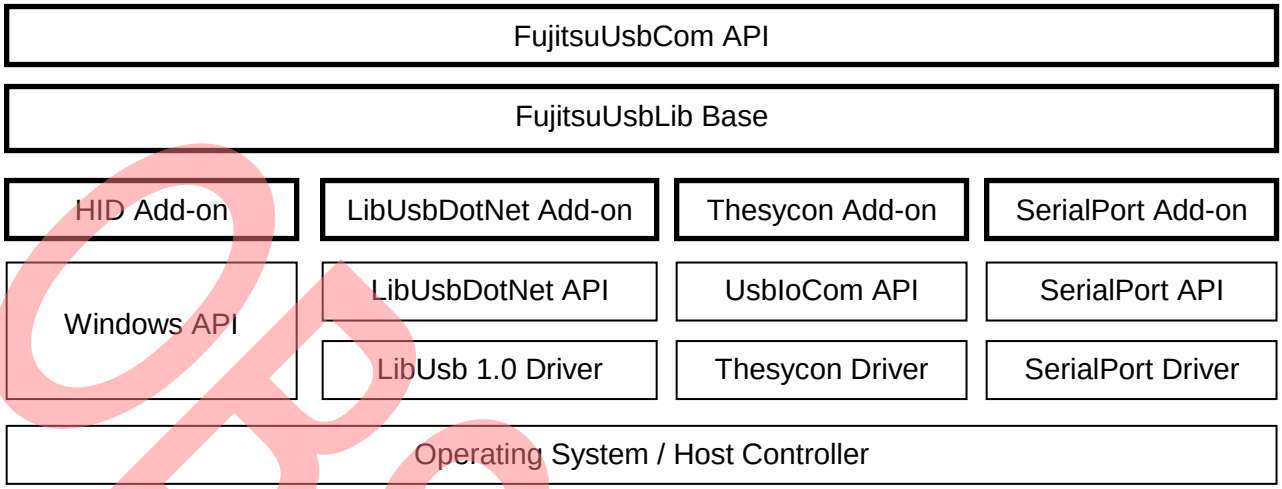
1 Introduction

For the most developers USB is not easy to use. Before USB, applications used the serial port to transfer data between embedded device and the personal computer. Today most personal computers do not have a serial port anymore. To build a bridge between the missing serial port and the embedded device, solutions from FTDI for example were used to convert USB to serial. But this is still a serial protocol and has the need of a second chip in design. Fujitsu provides also MCUs with embedded USB. For switching between different transfer protocols and interfaces, the FujitsuUsbLib was developed. The library also supports com ports, so the same API can be used for “old” and new devices.

The FujitsuUsbLib is programmed for Microsoft Windows .NET applications. It has backends for the serial port, LibUsbDotNet, HID and the Thesycon UsbIoCom. At the end different driver types can be used with only one API. The usage is as simple as the serial com port object in Visual Studio .NET.

The FujitsuUsbLib is only allowed to use with Fujitsu hardware. For all other purposes it is NOT allowed to use this library!

Following diagram shows the structure of the FujitsuUsbLib. The FujitsuUsbLib consists of three main categories: FujitsuUsbCom, FujitsuUsbLib Base and FujitsuUsbLib Add-On. The FujitsuUsbCom represents the API. This API is always the same. The FujitsuUsbLib Base handles necessary classes which are adopted by the FujitsuUsbLib Add-on.



1.1 Requirements

1.1.1 Operation System

Windows 2000, XP (32/64-Bit), Vista (32/64-Bit) or 7 (32/64-Bit)

1.1.2 .NET Runtime

Minimum version 2.0, optimum version 3.5

(for LibUsb usage min. version 3.5)

1.2 LibUsbDotNet

LibUsbDotNet is open source a can be found at http://libusbdotnet.sourceforge.net. It is provided under GPL and GPL2. For this reason the FujitsuUsbLib is partly code opened. The HID addon and the Thsycon addon is not part of the LibUsbDotNet routines.

LibUsbDotNet uses LibUsb 1.0 drivers for Microsoft Windows systems. Supported systems are Windows 2000 up to Windows 7 64-Bit.

1.3 Thesycon

Thesycon develops USB drivers for Windows and Embedded. For 16FX and FR MCUs an embedded USB device stack is available. To communicate with it, the Thesycon PC drivers can be used. For this reason also the Thesycon UsbIoCom API is implemented in backend.

1.4 Fujitsu HID

The Fujitsu HID addon adds different HID communication functionality. The Fujitsu HID addon is part of the FujitsUsbLib and is only allowed to use with Fujitsu hardware. For all other purposes it is NOT allowed to use this library!

1.5 SerialPort

As serial port normal ports and virtual com ports can be used.

1.6 Embedded USB Device Library

For using an API on embedded side, see also application note:
mcu-an-300132 (embedded usb device library).

2 FujitsuUsbCom Module

The acronym FujitsuUsbCom module stands for Fujitsu USB Communication Module. It helps choosing the driver backend API and handling this API. At the end the API for the user is always the same. It does no matter which driver was chosen in the backend.

The FujitsuUsbCom module handles data transfers, device connection and disconnection, device plug & play, different events and USB easy to go.

2.1 Fields

2.1.1 FujitsuUsbCom.AutoOpen

Device will be automatically opened/closed during Plug 'n' Play

Definition

```
public bool AutoOpen;
```

2.1.2 FujitsuUsbCom.DeviceProductID

USB Device Product ID

Definition

```
public int DeviceProductID
```

2.1.3 FujitsuUsbCom.DeviceVendorID

Vendor ID. The initial ID is 0x1F55 Fujitsu Semiconductor Europe Vendor ID

Definition

```
public int DeviceVendorID
```

2.1.4 FujitsuUsbCom.DriverType

Type of driver used to communicate

Definition

```
public FujitsuUsbLib.Driver.USB_DRIVER DriverType
```

2.2 Methods

2.2.1 FujitsuUsbCom.Close()

This method is used to close a opened device. If this method is called, the AutoOpen flag is automatically cleared.

Definition

```
public bool Close();
```

Returns

TRUE if the operation was succesfull

2.2.2 FujitsuUsbCom.DisableReceivedQueue(int)

This method is used to disable the received data queue for the specified endpoint. Data queues are used to store data after a received event. The method `GetReceivedSize` can be used to get the current data length stored in the queue. Calling `FlushQueue` flushes all data in the queue. This operation is only available for IN direction endpoints.

Definition

```
public bool DisableReceivedQueue(int EndpointNumber);
```

Parameter

`EndpointNumber` The Endpoint using the queue

Returns

TRUE if the operation was successful

2.2.3 FujitsuUsbCom.EnableReceivedQueue(int)

This method is used to enable the received data queue for the specified endpoint. Data queues are used to store data after a received event. The method `GetReceivedSize` can be used to get the current data length stored in the queue. Calling `FlushQueue` flushes all data in the queue. This operation is only available for IN direction endpoints.

Definition

```
public bool EnableReceivedQueue(int EndpointNumber);
```

Parameter

`EndpointNumber` The Endpoint using the queue

Returns

TRUE if the operation was successful

2.2.4 FujitsuUsbCom.FlushQueue(int)

This method is used to flush the received data queue for the specified endpoint. Data queues are used to store data after a received event. The method `GetReceivedSize` can be used to get the current data length stored in the queue. This operation is only available for IN direction endpoints. Before data queues can be used they have to be enabled via `EnableReceivedQueue`.

Definition

```
public bool FlushQueue(int EndpointNumber);
```

Parameter

`EndpointNumber` The Endpoint using the queue

Returns

TRUE if the operation was successful

2.2.5 FujitsuUsbCom.GetReceivedSize(int)

This method returns the received data length in queue for the specified endpoint. Data queues are used to store data after a received event. The method `GetReceivedSize` can be used to get the current data length stored in the queue. Calling `FlushQueue` flushes all data in the queue. This operation is only available for IN direction endpoints. Before data queues can be used they have to be enabled via `EnableReceivedQueue`.

Definition

```
public int GetReceivedSize(int EndpointNumber);
```

Parameter

`EndpointNumber` The Endpoint using the queue

Returns

The length of data which was received (stored in data queue)

2.2.6 FujitsuUsbCom.IsOpen()

Definition

```
public bool IsOpen()
```

Return Value

TRUE if it device is open

2.2.7 FujitsuUsbCom.IsValid()

This method validates if the driver handler is valid.

Definition

```
public bool IsValid()
```

Return Value

TRUE if it valid

2.2.8 FujitsuUsbCom.Open()

Used to open the USB device

Definition

```
public bool Open()
```

Return Value

TRUE if the device was opened successfully.

2.2.9 FujitsuUsbCom.Receive(int, out byte[], int)

Used to open the USB device

Definition

```
public bool Receive(
                    int EndpointNumber,
                    out byte[] Buffer,
                    int Timeout
                   );
```

Parameter

EndpointNumber	Endpoint for receiving data
Buffer	Byte array containing data
Timeout	Timeout in ms

Return Value

TRUE if the data was received successfully.

Note:

For HID communications the transferred buffers are 64 bytes! As endpoint any port can be specified because only one endpoint pair is specified. The driver automatically chooses the right endpoint for receiving data.

2.2.10 FujitsuUsbCom.Send(int byte[], int)

Used to open the USB device

Definition

```
public bool Send(
                        int EndpointNumber,
                        byte[] Buffer,
                        int Timeout
                );
```

Parameter

EndpointNumber	Endpoint for sending data
Buffer	Byte Array containing data
Timeout	Timeout in ms

Return Value

TRUE if the data was sent successfully.

Note:

For HID communications the transferred buffers are max. 64 bytes! As endpoint any port can be specified because only one endpoint pair is specified. The driver automatically chooses the right endpoint for sending data.

2.3 Events

2.3.1 FujitsuUsbCom.DeviceConnected

Event Handler for detecting a device connection

Definition

```
public event System.EventHandler<EventUsbDeviceArrival> DeviceConnected;
```

2.3.2 FujitsuUsbCom.DeviceDisconnected

Event Handler for detecting a device disconnection

Definition

```
public event System.EventHandler<EventUsbDeviceRemoval> DeviceDisconnected;
```

2.3.3 FujitsuUsbCom.OnDataTransferred

Event Handler for Data Transferred (Received)

Definition

```
public event System.EventHandler<DataTransferredEvent> OnDataTransferred;
```

2.4 C# Examples

2.4.1 Adding References

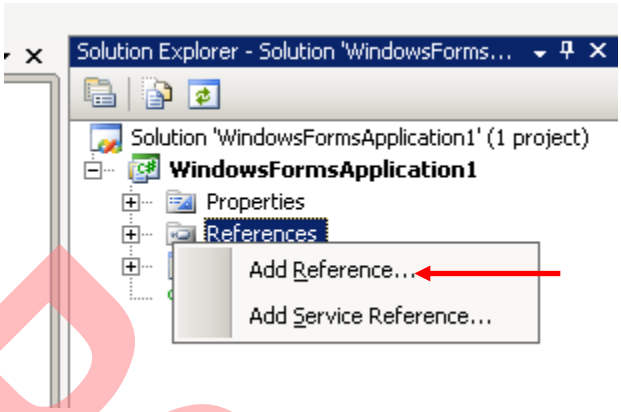
For using the Fujitsu Semiconductor Europe USB Communications Library following files are needed:

Library (Dll)	Description	Needed for Driver API			
		HID	LibUsb	Thesycon	Serial
FujitsuUsbLib.dll	Fujitsu USB Library (needed)	x	x	x	x
FujitsuUsbLibBase.dll	Fujitsu USB Library (needed)	x	x	x	x
FujitsuUsbLib_HidAddon.dll	Fujitsu HID support (optional)	x			
FujitsuUsbLib_LibUsbDotNetAddon.dll	LibUsbDotNet support (optional)		x		
FujitsuUsbLib_ThesyconAddon.dll	Thesycon driver support (optional)			x	
FujitsuUsbLib_SerialPortAddon.dll	Serial Port support (optional)				x
LibUsbDotNet.dll	LibUsbDotNet API library (optional)		x		
Interop.USBIOCOMLib.dll	Thesycon API library (optional)			x	
Supported Operating Systems					
Windows 2000		Yes	Yes	Yes	Yes
Windows XP (32-Bit)		Yes	Yes	Yes	Yes
Windows XP (64-Bit)		Yes	Yes	Yes	Yes
Windows Vista		Yes	Yes	Yes	Yes
Windows 7 (32-Bit)		Yes	Yes	Yes	Yes
Windows 7 (64-Bit)		Yes	Yes	Yes	Yes
Driver Needed		No	Yes	Yes	*)

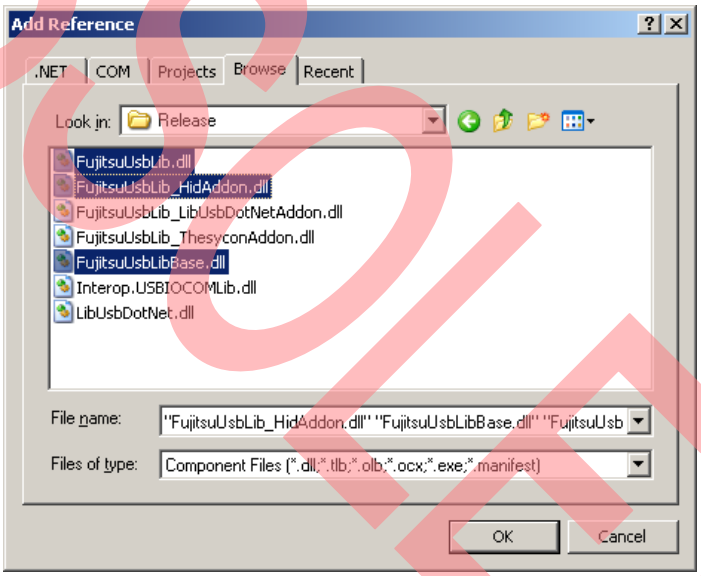
*) Depends on the serial port. For virtual com ports for example, drivers are needed.

2.4.2 Adding References for HID usage in C#

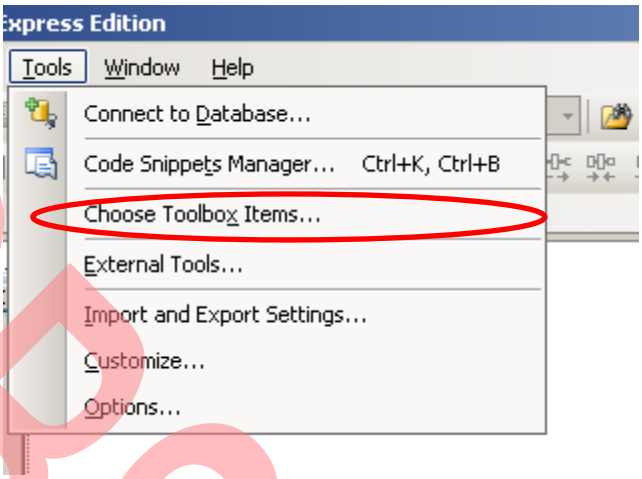
Add in the Solution Explorer a reference by right-clicking at References. Choose “Add Reference”.



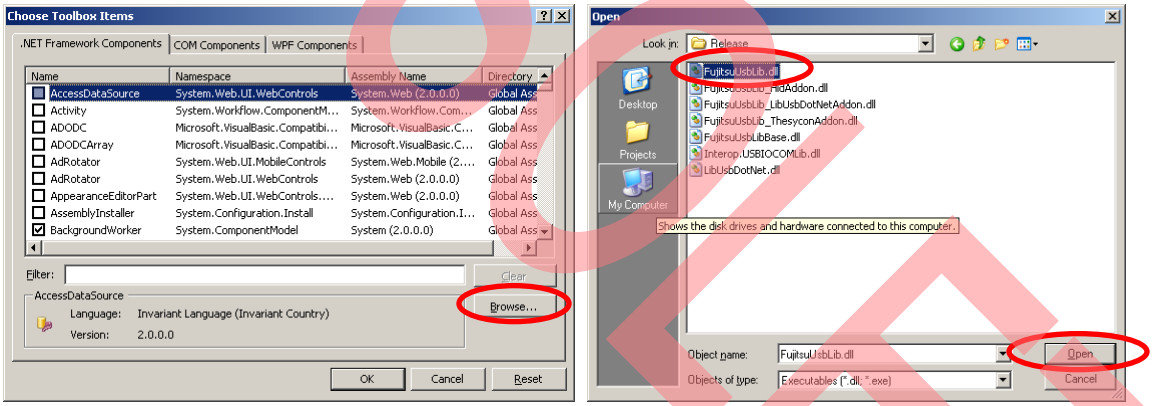
Add the libraries needed for HID USB communications:



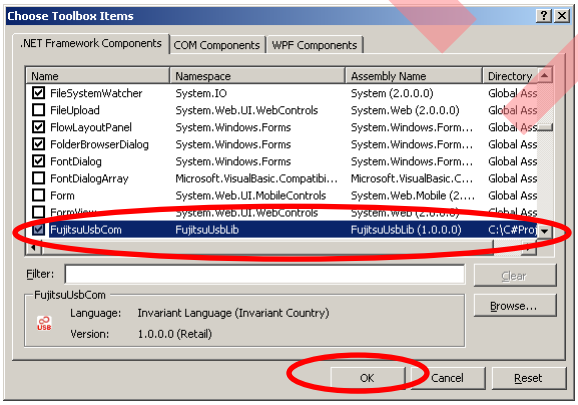
For the design backend also a component exists. This component can be easily added into the tool box. Go Tools->Choose Toolbox Items...



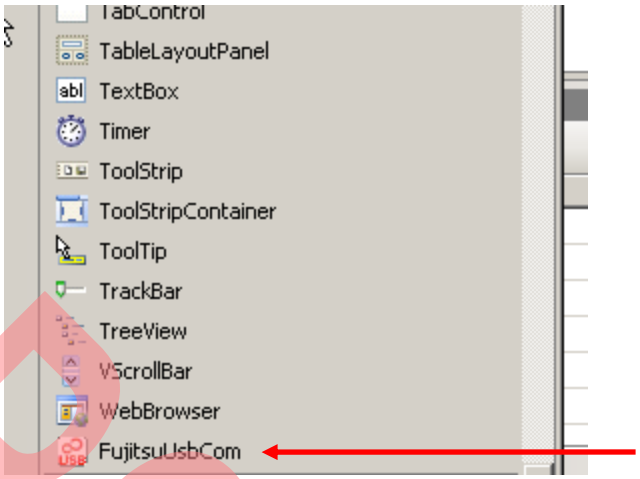
Browse the FujitsuUsbLib.dll .NET component:



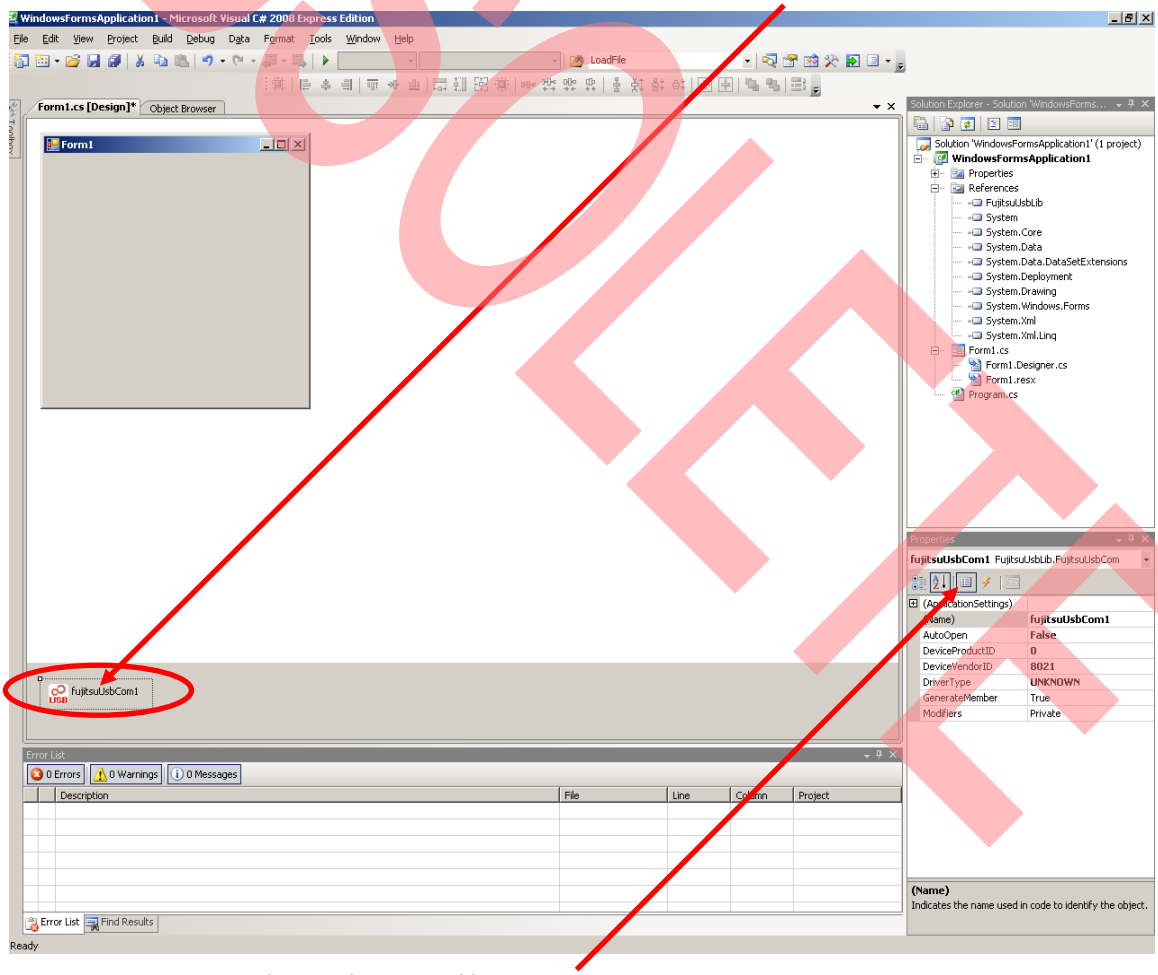
Select the library and click OK.



Now the Fujitsu USB Communications component can be used.



After dragging the component into the design frontend it will be displayed.



For usage, some properties can be entered here.

AutoOpen:

This can be used to auto-open the device after connect. For different reasons it is better to call this routine in the initialisations section of your application. The auto-open option should be normally set after the open command. See also example: Open Device.

DeviceProductID: Enter here your USB Device Product ID

DeviceVendorID: Enter here your USB Device Vendor ID

DriverType: Choose here your kind of driver. This can be HID, LibUsbDotNet or Thesycon.

Now your configuration is done.

2.4.3 Open Device

```
FujitsuUsbCom fujitsuUsbCom1 = new FujitsuUsbCom();
private void Example_DeviceOpen()
{
    //first step: Set Vendor ID and Product ID
    fujitsuUsbCom1.DeviceVendorID = 0x1F55;
    fujitsuUsbCom1.DeviceProductID = 0x000A;

    //second step: Choose the driver type
    fujitsuUsbCom1.DriverType = FujitsuUsbLib.Driver.USB_DRIVER.FUJITSU_HID;

    //third step: Open device
    fujitsuUsbCom1.Open();
}
```

2.4.4 Open Device as Serial Port

```
FujitsuUsbCom fujitsuUsbCom1 = new FujitsuUsbCom();
private void Example_DeviceOpen()
{
    //first step: Set Vendor ID and Product ID
    fujitsuUsbCom1.DeviceVendorID = 0xFFFF;  // serial port has no IDs…
    fujitsuUsbCom1.DeviceProductID = 0xFFFF; // serial port has no IDs…

    //second step: Choose the driver type
    fujitsuUsbCom1.DriverType = FujitsuUsbLib.Driver.USB_DRIVER.FUJITSU_HID;

    //third step: Set custom driver settings: Com Port 1, Baudrate 9600
    fujitsuUsbCom1.DriverSettings =
          new FujitsuUsbLib.Driver.SpecialSerialPortSettings(1, 9600, null, null);

    //forth step: Open device
    fujitsuUsbCom1.Open();
}
```

2.4.5 Open Device as Serial Port with Magic Words

Magic words are byte arrays defining start of block and end of block. This is implemented in the serial port add-on. If magic start and magic stop are not null, the add-on automatically returns only blocks. For switching from serial port to USB, it is better to implement a block transfer protocol to the serial port. The magic word for start should not be the same as for stop. In following example, following sequences are used for magic start: 0xFF, 0xFF, 0x00, 0x00, 0x01, 0x55, 0x55 and for magic stop: 0xFF, 0xFF, 0x00, 0x00, 0x02, 0xAA, 0xAA

```
FujitsuUsbCom fujitsuUsbCom1 = new FujitsuUsbCom();
private void Example_DeviceOpen()
{
    //first step: Set Vendor ID and Product ID
    fujitsuUsbCom1.DeviceVendorID = 0xFFFF;  // serial port has no IDs…
    fujitsuUsbCom1.DeviceProductID = 0xFFFF; // serial port has no IDs…

    //second step: Choose the driver type
    fujitsuUsbCom1.DriverType = FujitsuUsbLib.Driver.USB_DRIVER.FUJITSU_HID;

    //third step: Set custom driver settings: Com Port 1, Baudrate 9600
   Byte[] mStart = { 0xFF, 0xFF, 0x00, 0x00, 0x01, 0x55, 0x55 };
    Byte[] mStop = { 0xFF, 0xFF, 0x00, 0x00, 0x02, 0xAA, 0xAA };
    fujitsuUsbCom1.DriverSettings =
        new FujitsuUsbLib.Driver.SpecialSerialPortSettings(1, 9600, mStart, mStop);

    //forth step: Open device
    fujitsuUsbCom1.Open();
}
```

2.4.6 Send Data

```
private void Example_Send()
{
    //inital data to send
    Byte[] MyData = { (Byte)'T', (Byte)'e', (Byte)'s', (Byte)'t' };

    //send data via endpoint 1 with 1000ms timeout
    fujitsuUsbCom1.Send(1, MyData, 1000);
}
```

Note:

For HID communications the transferred buffers are max. 64 bytes! As endpoint any port can be specified because only one endpoint pair is specified. The driver automatically chooses the right endpoint for sending data.

2.4.7 Receive Data

```
private void Example_Receive()
{
    //inital data to send
    Byte[] MyData;

    Boolean Result = false;
    //while data was not received
    while (Result == false)
    {
        Result = fujitsuUsbCom1.Receive(2, out MyData, 1000);
    }
    //data was received to array MyData
}
```

Note:

For HID communications the transferred buffers are 64 bytes! As endpoint any port can be specified because only one endpoint pair is specified. The driver automatically chooses the right endpoint for receiving data.

2.4.8 Receive Data via Event

```
FujitsuUsbCom fujitsuUsbCom1 = new FujitsuUsbCom();
private void Example_DeviceOpen()
{
    //first step: Set Vendor ID and Product ID
    fujitsuUsbCom1.DeviceVendorID = 0x1F55;
    fujitsuUsbCom1.DeviceProductID = 0x000A;

    //second step: Choose the driver type
    fujitsuUsbCom1.DriverType = FujitsuUsbLib.Driver.USB_DRIVER.FUJITSU_HID;

    //third step: Adding received event handler
    fujitsuUsbCom1.OnDataTransferred += ReceviedEvent;

    //third fourth: Open device
    fujitsuUsbCom1.Open();
}

private void ReceviedEvent(object sender, FujitsuUsbLib.Events.DataTransferredEvent e)
{
    Byte[] MyData = e.Buffer;
    /* Data in MyBuffer can be used now */
}
```

Note:

For HID communications the transferred buffers are 64 bytes! As endpoint any port can be specified because only one endpoint pair is specified. The driver automatically chooses the right endpoint for receiving data.

2.4.9 Receiving Data via Queue

```
private void Example_StartQueueTransfers()
{
    fujitsuUsbCom1.EnableReceivedQueue(2); // Start received queue for endpoint 2
    Thread ReceiveThread =
        new Thread(new ParameterizedThreadStart(Example_CheckReceivedDataQueue));
    ReceiveThread.IsBackground = true; // closes thread after app ws closed
    ReceiveThread.Start();
}

private void Example StopQueueTransfers()
{
    fujitsuUsbCom1.DisableReceivedQueue(2); // Stop received queue for endpoint 2
}

private void Example_CheckReceivedDataQueue()
{
    Byte[] MyBuffer;
    while (fujitsuUsbCom1.GetReceivedSize(2) == -1) // while queue is enabled
    {
        while (fujitsuUsbCom1.GetReceivedSize(2) > 0) // data packages are in queue
        {
            fujitsuUsbCom1.Receive(2, out MyBuffer, 1000);
            /* Data in MyBuffer can be used now */
        }
        Thread.Sleep(1000); // sleep 1 second
    }
    fujitsuUsbCom1.FlushQueue(2); // flush queue before killing this thread
}
```

Note:

For HID communications the transferred buffers are 64 bytes! As endpoint any port can be specified because only one endpoint pair is specified. The driver automatically chooses the right endpoint for receiving data.

3 Appendix

3.1 List of litrature

Jan Axelson:

USB Complete - Everything You Need to Develop Custom USB Peripherals

Jan Axelson – USB Central

http://www.lvr.com/usb.htm

USB Implementers Forum (www.usb.org)

USB 2.0 Specification (http://www.usb.org/developers/docs)

LibUsb

http://www.libusb.org

LibUsbDotNet

http://libusbdotnet.sourceforge.net

Thesycon

http://www.thesycon.de

USB in a NutShell

http://www.beyondlogic.org/usbnutshell/

USB made simple

http://www.usbmadesimple.co.uk/

4 Document History

Document Title: AN205489 - F²MC-16FX/FR/FM3 USB Family 16/32-Bit USB Microcontroller USB MCU Series Fujitsu USB Lib

Document Number: 002-05489

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	-	MSCH	07/21/2011	MSC, V1.0 – First Version
*A	5292004	MSCH	06/24/2016	Updated with new template
*B	5612304	WOFR	01/31/2017	Spec obsoleted, no further updates planned.

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

ARM® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Lighting & Power Control	cypress.com/powerpsoc
Memory	cypress.com/memory
PSoC	cypress.com/psoc
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless/RF	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP

Cypress Developer Community

Forums | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com