Document name: EC-USB Getting started guide	Version 2.0.3
Internal reference: Products/EC-USB/Getting started guide/2768	

EC-USB Getting started guide

Version 2.0.3

Table of contents

1 Introduction

The EmbCode USB package (EC-USB) is a package of embedded drivers, windows drivers, windows software and documentation designed to help developers of embedded systems with USB support.

This document is intended for developers that are about to start using EC-USB.

Section 2 Overview describes how EC-USB works and how it relates to the rest of your application.

Section 3 Planning use of EC-USB guides you through the decisions you need to make before adding EC-USB to your code.

Section 4 Using EC-USB in your application is a practical guide on how to add the EC-USB code to your application.

Section 5 Windows drivers contains a guide of how to modify, build and sign the Windows drivers.

2 Overview

2.1 Software layers in a complete system using EC-USB

In order to better understand how EC-USB fits in the rest of your application code the image below might be helpful.

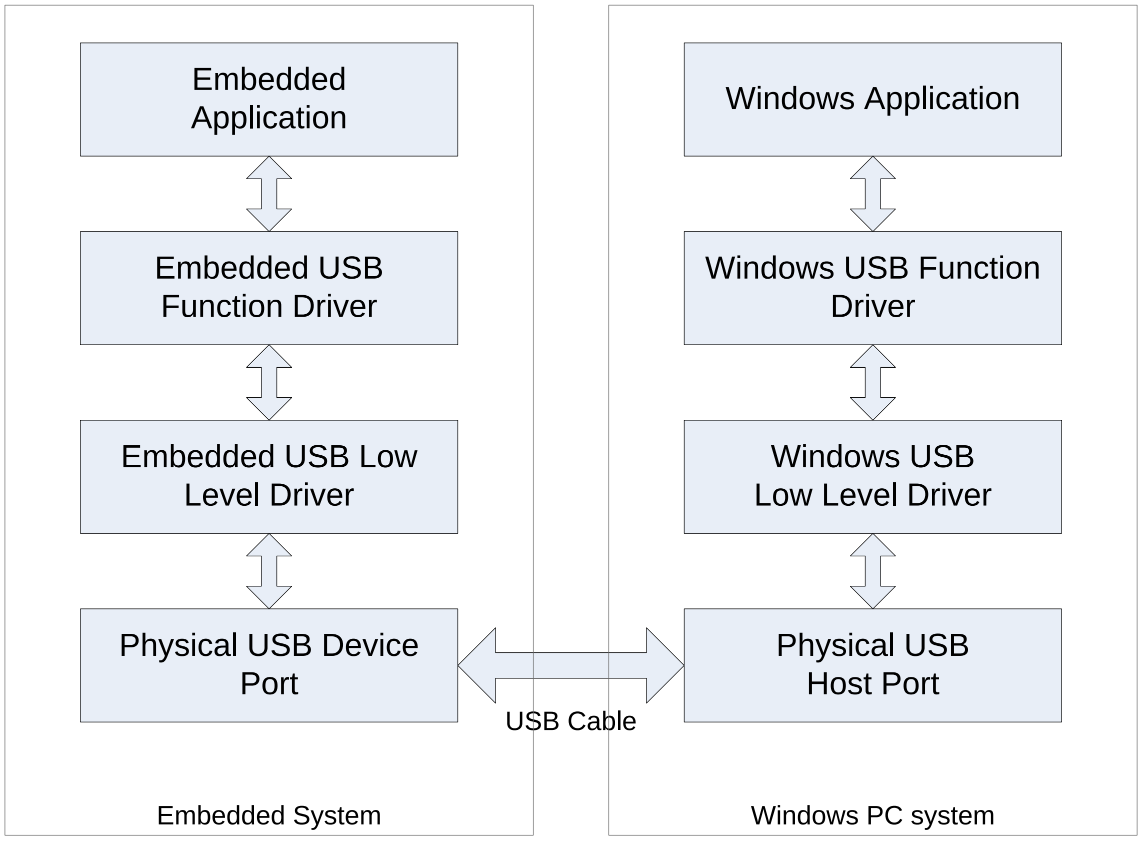


Figure 1

EC-USB contains both software for use on your embedded target and for use on your Windows system.

Embedded Application:
This is the main application code for the embedded device. It will send/receive data to the Windows application.

Embedded USB Function Driver:
The part of the embedded USB driver that implements the high level function of the USB device. This is for instance a function driver that implements all the interfaces necessary for a serial port.

USB Low Level Driver:
This is the low level driver that interfaces directly with the hardware. It is called by, and calls, the function driver. It is resposible for receiving and sending packets in the USB hardware.

Physical USB Device Port:
This is the physcial USB port that connect to the PC.

Windows Application:
The main application on the Windows PC. It will send/receive data to the Embedded application.

Windows USB Function Driver:
This is normally just called a Windows Driver. It uses the interfaces provided by the USB Device and presents a interface on the Windows computer which applications can use.

Windows Low Level Driver:
The part of the Windows Operating system that communicates with the the USB host port. This is included with Windows or your USB controller card so you will not need to write it yourself.

Physical USB Host Port:
The physical USB host port that connect to the embedded device port.

2.2 Drivers and samples included in EC-USB

The image below depicts which drivers and samples are included in EC-USB. It should be compared to Figure 1 and has the same layout.

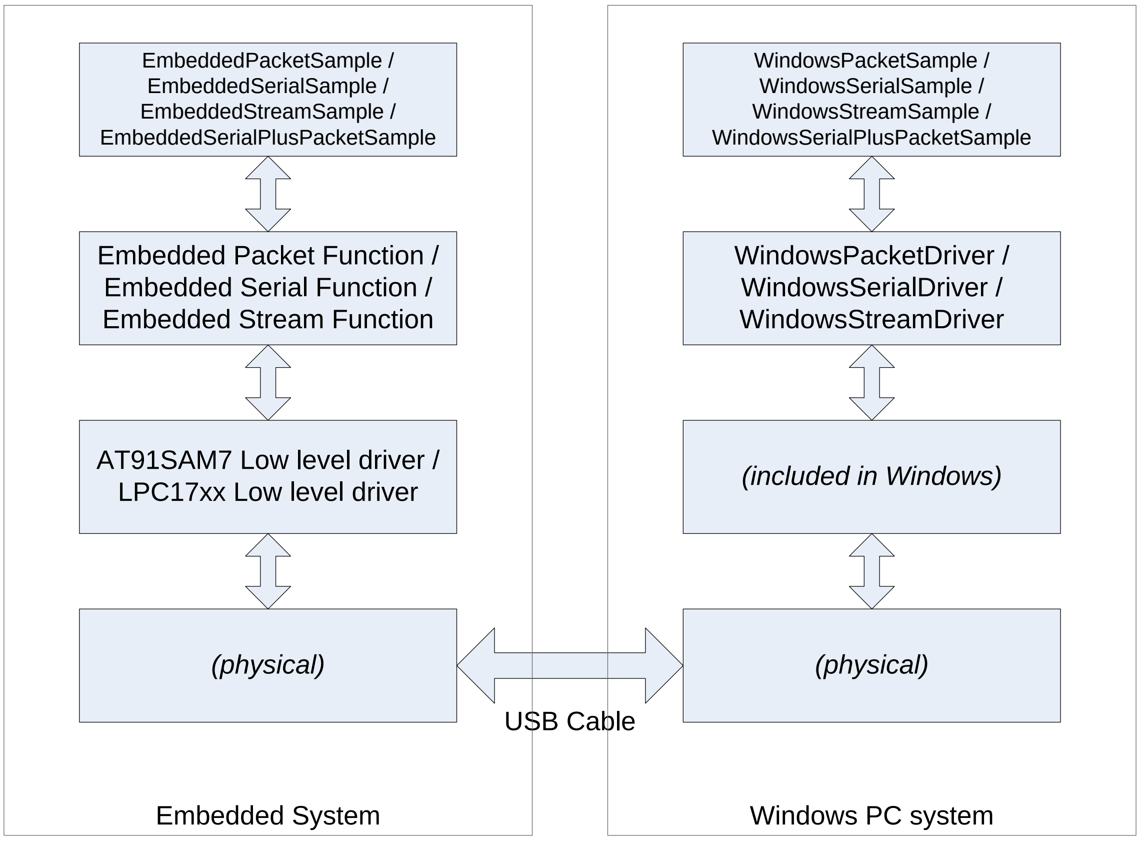

EmbeddedPacketSample / EmbeddedSerialSample / EmbeddedStreamSample / EmbeddedSerialPlusPacketSample
Embedded Packet Function / Embedded Serial Function / Embedded Stream Function
AT91SAM7 Low level driver / LPC17xx Low level driver
(physical)
Embedded System
USB Cable
WindowsPacketSample / WindowsSerialSample / WindowsStreamSample / WindowsSerialPlusPacketSample
WindowsPacketDriver / WindowsSerialDriver / WindowsStreamDriver
(included in Windows)
(physical)
Windows PC system

Figure 2

2.3 EC-USB directory structure

The files you receive as part of EC-USB comes packaged in a directory structure like below:

```
EC-USB 2.0.3/
      Documentation/
      Embedded/
           Example Applications/
                IAR/
                     AT91SAM7S256/
                          ECUSBBoot/
                          EmbeddedPacketSample/
                          EmbeddedStreamSample/
                          EmbeddedSerialSample/
                          EmbeddedSerialPlusPacketSample/
                     LPC1768/
                          ECUSBBoot/
                          EmbeddedPacketSample/
                          EmbeddedStreamSample/
                          EmbeddedSerialSample/
                          EmbeddedSerialPlusPacketSample/
           Source/
                ECUSB/
                     Functions/
                          Packet/
                          Serial/
                          Stream/
                     LowLevel/
                          AT91SAM7/
                          LPC17xx/
      Windows/
           Drivers/
                Binary/
                     WindowsPacketDriver/
                     WindowsSerialDriver/
                     WindowsStreamDriver/
                     WindowsSerialPlusPacketDriver/
                     WindowsUSBBootloaderDriver/
                Source/
                     WindowsPacketDriver/
                     WindowsSerialDriver/
                     WindowsStreamDriver/
                     WindowsSerialPlusPacketDriver/
                     WindowsUSBBootloaderDriver/
           Example Applications/
                Binary/
                Source/
                     VS2005/
                          CPP/
                               UpdateFirmware/
                               WindowsPacketSampleCPP/
                               WindowsSerialSampleCPP/
                               WindowsStreamSampleCPP/
```

3 Planning use of EC-USB

3.1 General

EC-USB contains many different layers and it is usually not necessary to modify all the layers to get a working application.

Also keep in mind that if you do choose to modify the parts, you should modify and test only one part at a time.

3.2 Which driver type to choose

The low level drivers are specific for the CPU family you are using so choose the one that matches your CPU-family. Included are AT91SAM7 and LPC17xx.

3.2.1 Packet

For new applications, the Packet drivers and examples should be used. The Packet drivers allow you to send packets to and from your embedded devices.

Packets are easy to use, typically you would reserve the first byte of each packet to contain a BYTE that specifies which command you want to execute and then use the rest of the packet to contain parameters. The included EC-USB Bootloader is a good example of how to use the packet driver. See `EC-USB Bootloader.pdf`.

You set a maximum packet size in the configuration (by default 528 bytes).

3.2.2 Serial

If your application needs to appear to the host PC as a regular COM serial port, you should use the Serial drivers. It is a good migration route if you've previously used a USB to Serial dongle and are now using USB directly but don't want to update your PC applications just yet.

On the embedded side, you send data to and from a stream pipe and on the PC you can just open the COM port from your application or from a terminal program.

3.2.3 Stream

The Stream driver is very similar to the Serial driver but it does not look like a COM port on the host PC.

Use the Stream driver if you've previously used a USB to Serial dongle and don't want to modify your PC applications to use packet data instead but you want to stop user from using your device directly by opening it as a COM port.

3.3 Suggested development order

3.3.1 Include EC-USB in your embedded application

- Start by including the EC-USB software into your application. The embedded source is included in:

  ```
  EC-USB 2.0.3/Embedded/Source
  ```

  Copy these files to your global include path or to an include path of your project.

- Make the source files part of your build. Which files to include depend on which low level and function drivers you are using. If you were using the Packet driver on an LPC1768 you would include the files from:

  `ECUSB/LowLevel/LPC17xx/*.*` and
  `ECUSB/Functions/Packet/*.*` and
  `ECUSB/*.*`

  in your build.

- Create (or copy from the samples) a file called Project.h and place in in your application directory or somewhere in your include path. Project.h is included by all EC-USB files and is used to set options for EC-USB.

- Copy code from the Embedded samples that sends test data. If you are using the Packet drivers you would copy the examples from EmbeddedPacketSample.

- Test your application with the included signed Windows drivers and the pre-built sample binaries to check that your device can send an receive data without errors.

  *Note: There are two pre-built versions of each Windows driver included, the Free and the Checked build. In most cases you should **use the Free build**. The Free build is the Release version and the Checked build is the Debug version. The Checked build will only work on a special version of Windows called a Checked Windows build. If you install it on a normal (i.e. Free) Windows system, the driver will fail.*

3.3.2 Modify the Embedded and Windows application

- Once the tests work well, you can now start to modifiy the code on the embedded side in order to use it in your application.

- Make a Windows applilcation or integrate one of the Windows samples into your existing application. If you are using the Packet drivers you would use and modify:

  ```
  EC-USB 2.0.3/Windows/Example
  Applications/Source/VS2005/CPP/WindowsPacketSampleCPP/
  ```

At this point you have a working application that you may ship to customers and it's not necessary to do more modifications.

3.3.3 Change the driver name and ProductID (optional)

The next step is to modify the drivers to use your company name and to use one of the ProductIDs that has been assigned to you. For a guide of how to do this see section 5 Windows drivers.

3.3.4 Modify the Windows drivers to support specialised features

For some applications you might want to continue and modify the included Windows drivers but this should not be necessary for most applications.

4 Using EC-USB on the embedded system

4.1 Device with Packet function

4.1.1 Includes

You need to include the appropriate headers for you low level and function drivers. If you are using the packet driver and LPC17xx you will include these files.

```
#include "ECUSB/LowLevel/LPC17xx/ECUSB_LPC17xx.h"
#include "ECUSB/Functions/Packet/ECUSB_Packet.h"
```

Note, you do not need to include ECUSB/ECUSB.h. It is automatically included.

4.1.2 Data types

In order to use EC-USB you need a struct ECUSB_Device to hold state information about the USB device and at least one of the state structs for the function drivers. You do not need to place this in a custom structure but doing so makes your code easier to read and groups the related data.

If you are only using the ECUSB functions in one file you can make it using an unnamed struct like in the samples:

```
struct {
   struct ECUSB_Device           m_device;
   struct ECUSB_Packet_Function  m_packetFunction;
} myUsbDevice;
```

If you are using the ECUSB function across several files you instead need create a named stuct:

In one of you header (.h) files:

```
struct MyUsbDevice {
   struct ECUSB_Device           m_device;
   struct ECUSB_Packet_Function  m_packetFunction;
};

extern struct MyUsbDevice myUsbDevice;
```

In one of your source (.c) files:

```
struct MyUsbDevice myUsbDevice;
```

4.1.3 Initialize ECUSB

Before you can use ECUSB you need to initialize ECUSB and the low level driver.

```
if(ECUSB_Init(NULL) != ECUSB_SUCCESS)
   halt("Can't initialize ECUSB");
```

For the LPC17xx low level driver, configuration is usually not necessary so we can just pass NULL. If your low level driver requires configuration like the AT91SAM7 driver you need to create a struct ECUSB_AT91SAM7_Configuration, zero it, fill in the required parameters and pass it to ECUSB_Init(). See the EmbeddedPacketSample for AT91SAM7 for an example on how to do this.

4.1.4 Create/Initialize an USB Device

For each USB Device you use (which is in almost all cases just a single one) you need to initialize and configure your struct ECUSB_Device. This is done by zeroing a struct ECUSB_Device_Configuration, filling in your parameters and passing it to ECUSB_Device_Init().

```
// Initialize the USB Device
struct ECUSB_Device_Configuration deviceConfiguration;
memset(&deviceConfiguration, 0, sizeof(deviceConfiguration));

deviceConfiguration.m_wVendorID = 0x2351;
deviceConfiguration.m_wProductID = 0x0200;
deviceConfiguration.m_wDeviceReleaseMajor = 0;
deviceConfiguration.m_wDeviceReleaseMinor = 1;
deviceConfiguration.m_wPowerConsumption = 100;
deviceConfiguration.m_bSelfPowered  = FALSE;
deviceConfiguration.m_szManufacturer = "EmbCode.com";
deviceConfiguration.m_szProductDescription =
   "EmbCode.com USB Packet Sample Device";

if(ECUSB_Device_Init(&myUsbDevice.m_device, &deviceConfiguration)
      != ECUSB_SUCCESS)
   halt("ECUSB_Device_Init failed");
```

4.1.5 Initialize your functions

Once the device is initalized, you need to initialize all of the device functions. For this device we only have one packet function so we initialize it by filling in a struct ECUSB_Packet_Configuration and passing it to ECUSB_Packet_Init().

```
// Initialize the USB Packet function driver
struct ECUSB_Packet_Configuration packetConfiguration;
memset(&packetConfiguration, 0, sizeof(packetConfiguration));
packetConfiguration.m_pDevice = &myUsbDevice.m_device;

if(ECUSB_Packet_Init(
      &myUsbDevice.m_packetFunction,
      &packetConfiguration) != ECUSB_SUCCESS)

   halt("ECUSB_Packet_Init failed");
```

For most cases the function drivers just needs the struct ECUSB_Device as a configuration parameter.

This will initialize the struct ECUSB_Packet_Function that is used by the function driver to hold internal data. You will need to pass a pointer to this struct to each of the function drivers functions.

4.1.6 Attach your device

You are now ready to use the device and should attach the device to the host. Do this by calling ECUSB_Device_Attach()

```
if(ECUSB_Device_Attach(&myUsbDevice.m_device) != ECUSB_SUCCESS)
   halt("ECUSB_Device_Attach failed");
```

On a system with a configurable pull-up resistor, this will activate the pull-up resistor on D+ to indicate that the device is present and ready to communicate. On systems with a fixed pull-up resistor, you should attach your device as soon as possible since the host will only give your device a limited amount of time (usually 5 seconds) to respond before it declares that your device as faulty.

4.1.7 Wait for the host to load the drivers and application

Often it is preferable to now wait for the host application to be started and connect to the device. All the function drivers have a function called ECUSB_*_IsApplicationConnected(). This function is part of the function driver since there is no generic way of detecting this, it is specific to each function driver.

To wait for the host application to be started and connected the embedded can call ECUSB_Packet_IsApplicationConnected() for a packet function driver:

```
while(!ECUSB_Packet_IsHostConnected(&myUsbDevice.m_packetFunction));
```

If the embedded application tries to send data to the host when no host driver has been loaded or no host application has been started, the data will be dropped and lost.

4.1.8 Read data

For the packet driver you read data i two steps. First you acquire a pointer to the received packet via ECUSB_Packet_ReadPacket(). Once you are done processing the packet, you free the buffer for it by calling ECUSB_Packet_ReadPacketDone().

```
BYTE *packetData;
WORD packetSize;

for(;;) {
   if(ECUSB_Packet_ReadPacket(
         &myUsbDevice.m_packetFunction,
         &packetData,
         &packetSize) != ECUSB_SUCCESS)
      halt("ECUSB_Packet_ReadPacket() failed");

   // ...process packet in packetData of size packetSize...

   ECUSB_Packet_ReadPacketDone(&myUsbDevice.m_packetFunction);
}
```

ECUSB_Packet_ReadPacket() will block if there is no data available. If you want to check if a packet is available without blocking you need to call ECUSB_Packet_IsPacketAvailable.

4.1.9 Write data

Writing data is similar. You obtain a buffer to write to by calling ECUSB_Packet_WritePacket(). Once you are done writing the packet to the buffer, you call ECUSB_Packet_WritePacketDone() with the final packet size to actually send it.

```
BYTE *pPacketBuffer;
WORD wMaximumPacketSize;

if(ECUSB_Packet_WritePacket(&myUsbDevice.m_packetFunction,
           &pPacketBuffer, &wMaximumPacketSize) != ECUSB_SUCCESS)
   halt("ECUSB_Packet_WritePacket() failed");

// ...fill pPacketBuffer with data to send. Up to wMaximumPacketSize
// bytes...

if(ECUSB_Packet_WritePacketDone(
    &myUsbDevice.m_packetFunction,
    wActualPacketSize) != ECUSB_SUCCESS)
    halt("ECUSB_Packet_WritePacketDone() failed");
```

4.2 Composite Device with both Serial and Packet function

4.2.1 Data types

Creating a composite device is similar to creating a device with just one function. You will need to modify your custom struct to hold data for all of the device functions. For a device with both a serial and packet function you would use a struct like this:

```
struct {
   struct ECUSB_Device           m_device;
   struct ECUSB_Serial_Function  m_serialFunction;
   struct ECUSB_Packet_Function  m_packetFunction;
} myUsbDevice;
```

4.2.2 Initialize your functions

As with a single function device, you start be initializing ECUSB and the USB Device.

For a composite device, the function initialization is similar but you need to do it once for each function. So for a combined Serial + Packet device you need to call ECUSB_Serial_Init() and ECUSB_Packet_Init()..

```
// Initialize the USB Serial function driver
struct ECUSB_Serial_Configuration serialConfiguration;
memset(&serialConfiguration, 0, sizeof(serialConfiguration));
serialConfiguration.m_pDevice = &myUsbDevice.m_device;
```

```
   if(ECUSB_Serial_Init(
         &myUsbDevice.m_serialFunction,
         &serialConfiguration) != ECUSB_SUCCESS
      halt("ECUSB_Serial_Init failed");

   // Initialize the USB Packet function driver
   struct ECUSB_Packet_Configuration packetConfiguration;
   memset(&packetConfiguration, 0, sizeof(packetConfiguration));
   packetConfiguration.m_pDevice = &myUsbDevice.m_device;

   if(ECUSB_Packet_Init(
         &myUsbDevice.m_packetFunction,
         &packetConfiguration) != ECUSB_SUCCESS
      halt("ECUSB_Packet_Init failed");
```

Note that the order of initialization is important as it determines which order the interfaces of each function will be presented to the host.

All function drivers use one interface except for the serial function which uses two so after the above code the interfaces of the device will be:

Interface 0: Serial function base/master interface
Interface 1: Serial function secondary interface
Interface 2: Packet function base/master interface

The base/master interface is needed for the windows drivers to be able to connect properly to the device's function.

4.2.3 Reading/writing data

Reading and writing data to and from each function is now identical to doing so for a single function device.

5 Windows drivers

5.1 Modifying the drivername and ProductID

Start by copying the files from one of the drivers that you want to modify. If you are using the Packet drivers, you should copy the files from:

```
EC-USB 2.0.3/Windows/Drivers/Source/WindowsPacketDriver/
```

5.1.1 Project.h

Modify the file Project.h with a new GUID for the interface to your driver. We call this GUID the InterfaceGUID. GUID's can be generated with guidgen.exe that is included with Microsoft Visual Studio and WinDDK.

5.1.2 EmbCode_________.inf

Modify the .inf file (EmbCodeUsbPacket.inf for the Packet driver):

- Generate a new GUID for ClassGuid on line 10. We call this GUID the ClassGUID.
- Change the ProductID on line 27 and line 30 to your PID. (they are 0200 in the Packet drivers)
- If you are modifying the driver to use for a composite device, you will need to add "&MI_xx" to the end of line 27 and line 30 where xx is the master interface of the function that the driver should access. So if you are modifiying the packet driver for use in a packet function of a composite device with the master interface 2 you would add "&MI_02".
- Change Company, MediaDescription, DeviceDescription and ClassName at the end of the file (lines 94-97)

5.1.3 Modifiy the Windows Application

- Modify the Windows application you are using. You need to change the Interface GUID used to the new InterfaceGUID that we've generated. For the Packet drivers, the Interface GUID is defined in:

  ```
  EC-USB 2.0.3/Windows/Example
  Applications/Source/VS2005/CPP/WindowsPacketSampleCPP/Project.h
  ```

You should now check that the driver still builds and that it works with your application.

5.1.4 Modify filenames

You probably want to modify the file names of the included drivers. For the Packet driver you would change "EmbCodeUsbPacket" to the name of your driver.

- Rename EmbCodeUsbPacket.inf to YourDriver.inf
- Change the text "EmbCodeUsbPacket" to "YourDriver" in YourDriver.inf

- Change the text "EmbCodeUsbPacket" to "YourDriver" in build_all_targets.bat
- Change the text "EmbCodeUsbPacket" to "YourDriver" in build_all_targets.bat
- Change the text "EmbCodeUsbPacket" to "YourDriver" in sources

5.2 Building Windows drivers

The drivers are built using the Windows DDK which is free and available as a download from Microsoft. The included samples have been built using WinDDK 7600.16385.1.

If you install it in the default location of "C:\WinDDK\7600.16385.1" you can just run the included "build_all_targets.bat" to build the drivers. This will create a directory called Release.

The Release/Free folder contains the final version of your driver and is the one to distribute to end user.

The Release/Checked folder is a debug version that is used when debugging the driver.

Note that the path to the project you are building cannot contain spaces since it's not supported by WinDDK.

5.3 Signing Windows drivers

You will need to sign your drivers in order to install them to a Windows Vista or Windows 7 system.

Obtain a code signing certificate from a certificate authority. Make sure it is of the right type. E.g. for GlobalSign, their product is called "CodeSigning For Microsoft Authenticode".

When the Release-folder is built, it also contains a batch file named "sign_release.bat". Start this batch file to sign your driver.

In order to use it, you will need to set up the following ENV variables in your Windows system

- **CODESIGN_CERPATH**

  This is the path to the certificates that represents a path between your certificate and the root certificate from Microsoft. Your certificate authority will supply it for you.

  Example:
  SET CODESIGN_CERPATH=C:\mypath\MSCV-GlobalSign.cer

- **CODESIGN_P12PATH**

  This is the path to the P12 file that contains your certificate and your key. Your certificate authority will supply it for you once you obtain your signed certificate.

  Example:
  SET CODESIGN_P12PATH=C:\mypath\Embcode Authenticode.p12

- **CODESIGN_TIMESERVER**

  To get a signed timestamp and timeserver is needed. Your certificate authority will supply it for you.

Example:
SET CODESIGN_TIMESERVER=http://timestamp.globalsign.com/scripts/timstamp.dll

- **WINDDKDIR**

  The path to your WinDDK directory. If not set " C:\WinDDK\7600.16385.1" will be used.

  Example:
  SET WINDDKDIR=C:\WinDDK\7600.16385.1

5.4 Removing failing drivers

Occasionally when developing drivers, you will create a driver that hangs or crashes the system immediately when your device is inserted. In order to remove a failing driver do the following:

- Open a command prompt (make sure it has administrative privileges)
- Enter the following commands:

```
C:\>set devmgr_show_nonpresent_devices=1

C:\>start devmgmt.msc
```

- The Windows device manager will now open
- Select "View/Show hidden devices" from the menu.

You will now see all devices, even the ones that are not attached and you can right-click on them and select "Uninstall" to remove a failing driver.

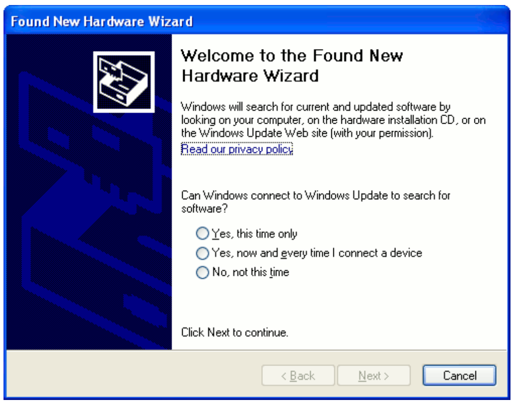
6 Appendix A: Driver installation and USB Serial number

When using the drivers for a USB device you have the choice to supply your device with a serial number or not. Depending on your decision, the user's experience of the driver installation will differ. This appendix details how driver installation is done in Windows and how it differs depending wheter you have given your device a serial number or not.

6.1 Windows XP

There are three main scenarios when plugging in a USB to a PC running Windows XP:

Manual installation (advanced): User is asked to search Windows Update. The user needs to direct the wizard to the driver files.



Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

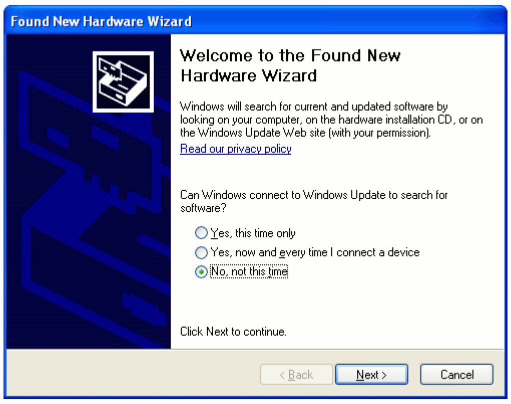
Read our privacy policy

Can Windows connect to Windows Update to search for software?

- Yes, this time only
- Yes, now and every time I connect a device
- No, not this time

Click Next to continue.

→



Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

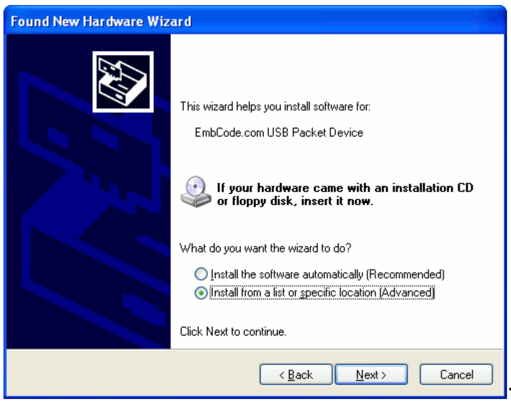
Read our privacy policy

Can Windows connect to Windows Update to search for software?

- Yes, this time only
- Yes, now and every time I connect a device
- [✓ No, not this time]

Click Next to continue.

→



This wizard helps you install software for:

EmbCode.com USB Packet Device

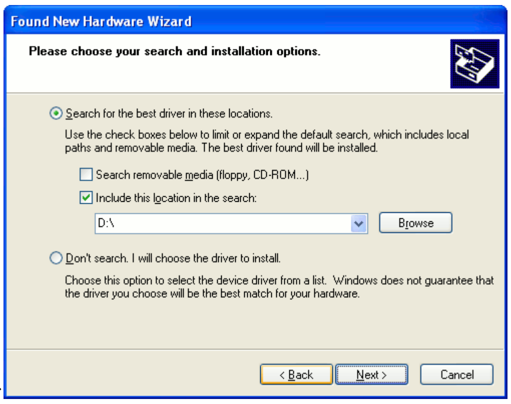
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

- Install the software automatically (Recommended)
- [✓ Install from a list or specific location (Advanced)]

Click Next to continue.

→



Please choose your search and installation options.

[✓ Search for the best driver in these locations.]

Use the check boxes below to limit or expand the default search, which includes local paths and removable media. The best driver found will be installed.

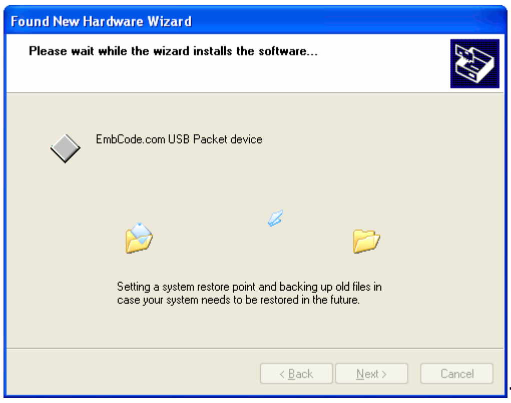
- Search removable media (floppy, CD-ROM...)
- [✓ Include this location in the search:]

D:\

Don't search. I will choose the driver to install.

Choose this option to select the device driver from a list. Windows does not guarantee that the driver you choose will be the best match for your hardware.

→

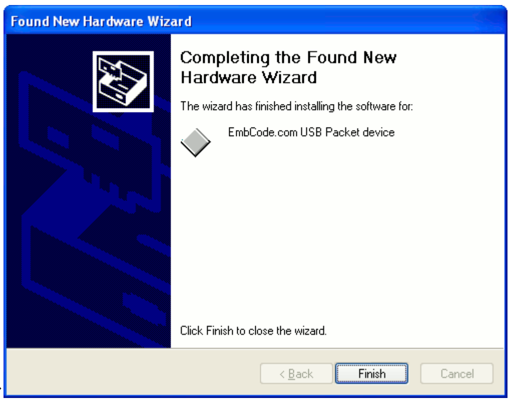

Found New Hardware Wizard

Please wait while the wizard installs the software...

EmbCode.com USB Packet device

Setting a system restore point and backing up old files in case your system needs to be restored in the future.

→


Found New Hardware Wizard

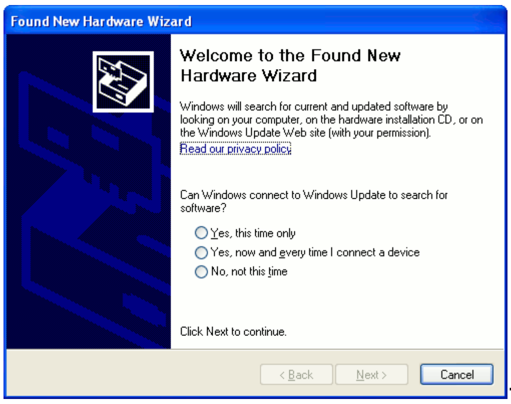
Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

EmbCode.com USB Packet device

Click Finish to close the wizard.

Manual installation (easy): User is asked for a driver but can click "Next >" through the wizard.


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

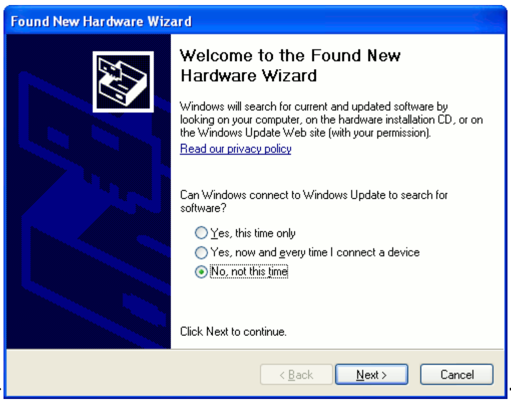
Read our privacy policy

Can Windows connect to Windows Update to search for software?

- Yes, this time only
- Yes, now and every time I connect a device
- No, not this time

Click Next to continue.

→


Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

Windows will search for current and updated software by looking on your computer, on the hardware installation CD, or on the Windows Update Web site (with your permission).

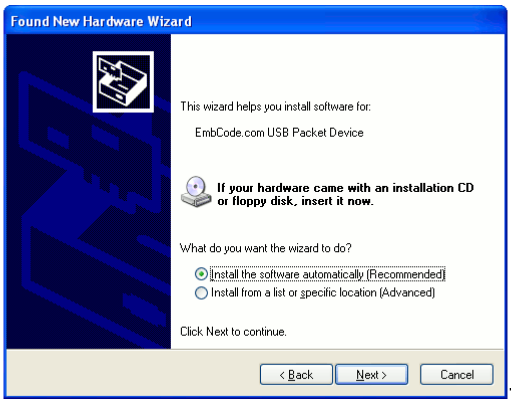
Read our privacy policy

Can Windows connect to Windows Update to search for software?

- Yes, this time only
- Yes, now and every time I connect a device
- [✓ No, not this time]

Click Next to continue.

→


Found New Hardware Wizard

This wizard helps you install software for:

EmbCode.com USB Packet Device

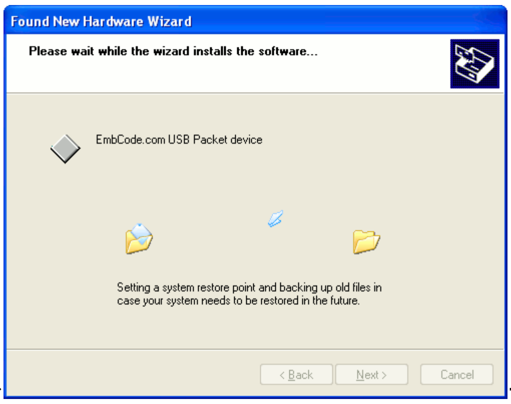
If your hardware came with an installation CD or floppy disk, insert it now.

What do you want the wizard to do?

- [✓ Install the software automatically (Recommended)]
- Install from a list or specific location (Advanced)

Click Next to continue.

→

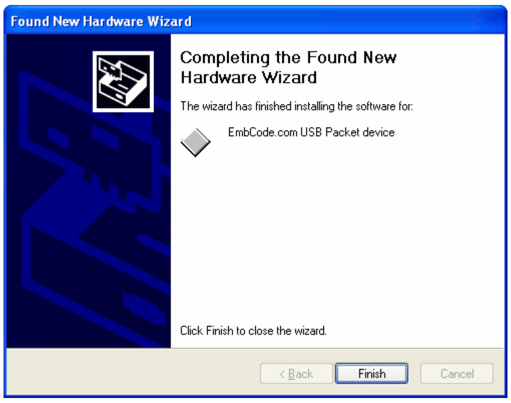

Found New Hardware Wizard

Please wait while the wizard installs the software...

EmbCode.com USB Packet device

Setting a system restore point and backing up old files in case your system needs to be restored in the future.

→


Found New Hardware Wizard

Completing the Found New Hardware Wizard

The wizard has finished installing the software for:

EmbCode.com USB Packet device

Click Finish to close the wizard.

No installation required: The device works immediately when inserted.

6.1.1 Behaviour in Windows XP

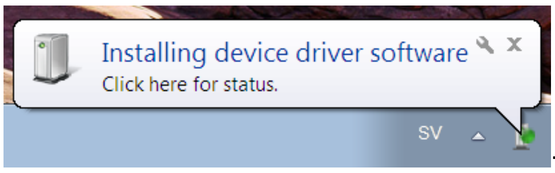
	Device **without** serial number	Device **with** serial number
First time a device with this VID/PID is plugged in into USB Port A	Manual installation (advanced)	Manual installation (advanced)
The device is moved from USB port A to USB Port B.	Manual installation (easy)	No installation required
A different device is plugged into USB Port A*	No installation required	Manual installation (easy)

* For a device without a serial number this means that the device will look identical to the PC. For a device with a serial number it is assumed that a different device will have a different serial number.

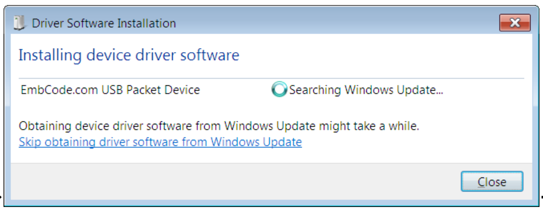
6.2 Windows 7

There are three main scenarios when plugging in a USB to a PC running Windows 7:

Manual installation (advanced): Windows will try to find a driver using Windows Update. When this fails, the user has to go to Device Manager to manually select a driver for the device.


Installing device driver software
Click here for status.

→

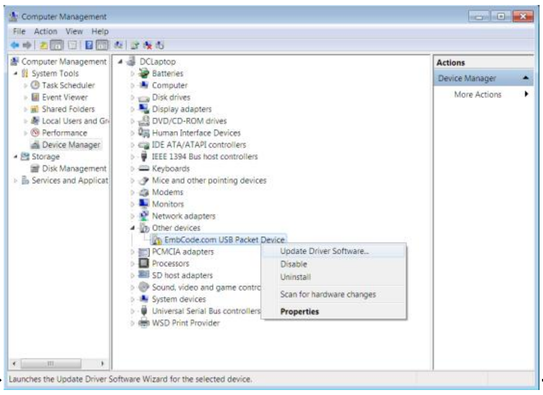

Driver Software Installation

Installing device driver software

EmbCode.com USB Packet Device Searching Windows Update...

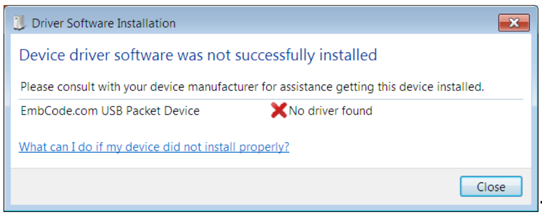
Obtaining device driver software from Windows Update might take a while.
Skip obtaining driver software from Windows Update

→


Computer Management

Launches the Update Driver Software Wizard for the selected device.

→


Driver Software Installation

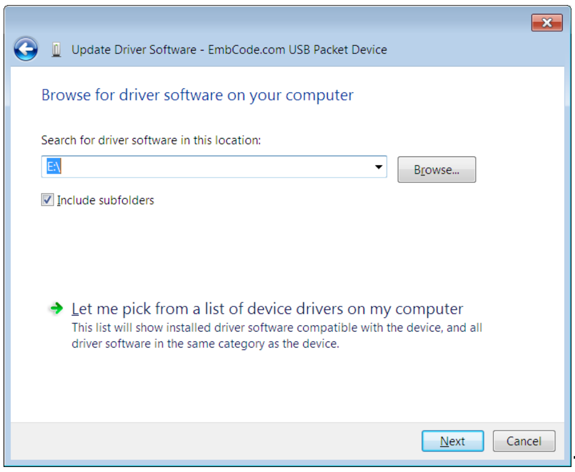
Device driver software was not successfully installed

Please consult with your device manufacturer for assistance getting this device installed.

EmbCode.com USB Packet Device ❌ No driver found

What can I do if my device did not install properly?

→


Update Driver Software - EmbCode.com USB Packet Device

Browse for driver software on your computer

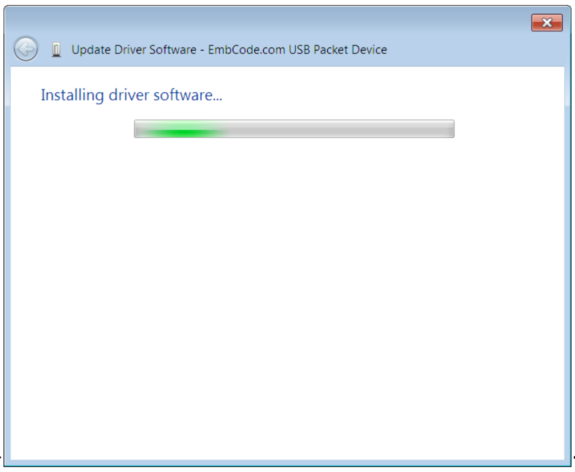
Search for driver software in this location:

E:\

☑ Include subfolders

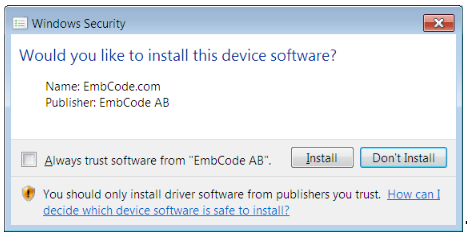
→ Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

→


Update Driver Software - EmbCode.com USB Packet Device

Installing driver software...

→

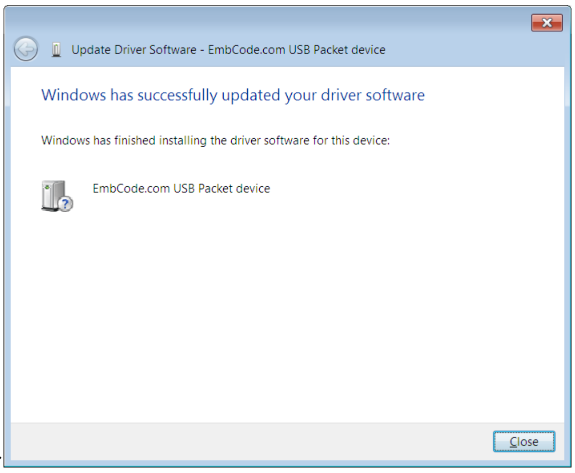


Windows Security

Would you like to install this device software?

Name: EmbCode.com
Publisher: EmbCode AB

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?



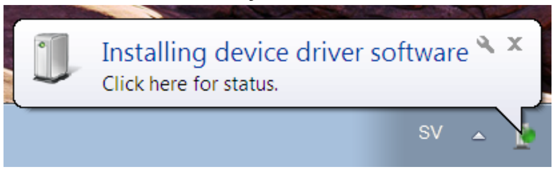
Update Driver Software - EmbCode.com USB Packet device

Windows has successfully updated your driver software

Windows has finished installing the driver software for this device:

EmbCode.com USB Packet device

Automatic installation: Windows will without user intervention install the driver for the device if the driver exists on the system. This will however be quite slow and take around 1 minute to complete.



Installing device driver software

Click here for status.

No installation required: The device works immediately when inserted.

6.2.1 Behaviour in Windows 7

	Device **without** serial number	Device **with** serial number
First time a device with this VID/PID is plugged in into USB Port A	Manual installation (advanced)	Manual installation (advanced)
The device is moved from USB port A to USB Port B.	Automatic installation	No installation required
A different device is plugged into USB Port A*	No installation required	Automatic installation

* For a device without a serial number this means that the device will look identical to the PC. For a device with a serial number it is assumed that a different device will have a different serial number.

6.3 Using the same serial number for several devices

The USB specification does in fact not specify that serial numbers need to be unique although the notion of a serial number implies that it should be. For some device classes, like the Mass Storage devices class, the specification actually requires a unique serial number.

It might be tempting to use the same serial number for several devices. This would in theory avoid both the driver installation when a device is moved from one USB port to another AND when a new device is connected.

This does actually work as long as you (and your users) never connect more than one device to aPC. If you connect two devices with the same serial number to a PC simultaneously, the result depends on several factors.

Window XP: Used to crash in early version but doesn't crash anymore.

It will work in most situations if the two devices are plugged into the same (root) hub but it will cause manual driver installation when the second device is attached.

But if connecting them to two different (root) hubs Windows will confuse the two devices and you won't be able to use both of them.

Windows 7: It will work but the second device connected will cause an automatic installation for each port it is connected to.

6.4 Conclusion

The first time a device is connected to the PC, the user will always need to direct Windows to find the drivers if they have not been pre-installed.

Device without serial number:
An automatic (Windows 7) or easy manual (Windows XP) driver installation is required every time a device is attached to a new USB port.

Device with serial number:
An automatic (Windows 7) or easy manual (Windows XP) driver installation is when a device with a **new** serial number is attached to any port.

Otherwise (for instance when moving the device to a new USB port) a driver installation is not needed.

6.5 Recommendations

For a bootloader the recommendation is to not use a serial number if you will use one computer to program many devices. If you use a serial number for the bootloader, each device will cause the driver to reinstall.

For other applications there isn't much difference between using a serial number and not using one. It is slightly easier for the user if the device has a unique serial number since the user then will be able to move the device between different USB ports without the driver reinstalling.

It is never advisable to use the same USB Serial number for several devices. Although it does appear to work in Windows 7, Microsoft officially requires the serial number to be unique if it is present so support is not guaranteed in future versions.