DEVCON

Enabling the Smart Society

OCTOBER 22-25, 2012

HYATT REGENCY ORANGE COUNTY

RENESAS

LibUSB - Create a Solution Without the Class Struggle

Renesas Electronics America Inc.

Renesas Technology & Solution Portfolio

Microcontroller and Microprocessor Line-up

2010 → 2012

32-bit

RENESAS SuperH

1200 DMIPS, Superscalar

- Automotive & Industrial, 65nm
- 600µA/

RENESAS RH850

1200 DMIPS, Performance

- ve, 40nm
- , 35µA deep standby

RENESAS V850

50

- Auto
- 600µ

MIPS, FPU, DSC

- 0nm
- , 0.3µA deep standby

RENESAS RX

16

- Indus
- 200µ

ed Security, ASSP

- 0nm
- 100µA standby

8/16-bit

RENESAS 78K

25

- Indust
- 190µA/MH

MIPS, True Low Power

- Industrial & Automotive, 130nm
- 144µA/MHz, 0.2µA standby

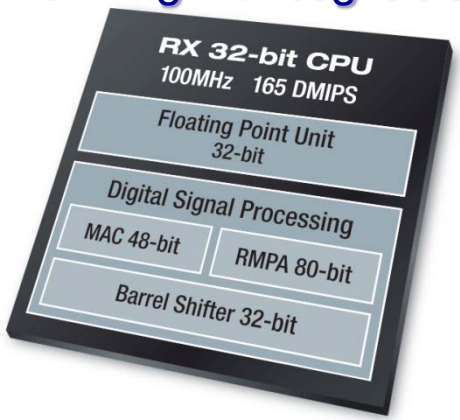
RENESAS R8C

10 DMIPS, Capacitive Touch

- Industrial & Automotive, 130nm
- 350µA/MHz, 1µA standby

RENESAS RX RX600

32-Bit High Performance DSP, FPU with High Integration



Connect to the Smart Society with USB!

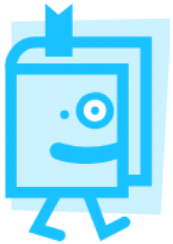
Smart Society Challenge

- You need to:
  - Get your idea to market SOON
  - Avoid the complexity of a class
  - Quickly try out a complete PC-host / target solution
  - Be “Vendor Specific” but don’t want to write a new Windows driver
- You don’t want to learn all about HID descriptors

  Etc..

Solution

- ***Use LibUSB!***
- Today you will get a complete USB
  Host-PC to **RX-function** ‘template’ solution

RX USB UNIVERSAL SERIAL BUS **+**  **=** 

Agenda

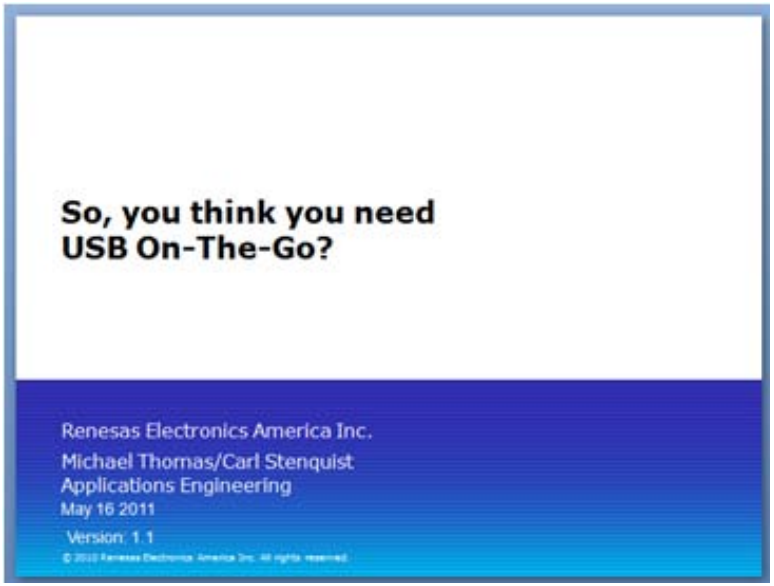
- I: **USB Basics**
  - Host
  - Function
  - Class
- II: **LibUSB**
  - Why use LibUSB?
  - What is LibUSB?
    - Win32-LibUSB
  - Enumeration, INF-file
  - Customer Install
- III: **Lab**
  - RX Target (YRDK63N) – PC Host / Python app
  - To do lab at home, download
    "LibUSB - A Complete RX Function and PC Host Solution", R01AN0492_RX600_LibUSBv2
- IV: **Conclusion**
  - How to communicate with *any* USB device with LibUSB

Ready-made USB Class Solutions

- Do follow a class when
  - Your solution naturally belongs there
    - Driver already exists in Windows
  - Industry practise mandates

- Renesas has available source code for classes
  - Mass Storage
    - HMSC
    - PMSC
  - Communications Device
    - HCDC
    - PCDC
  - Human Interface
    - HHID
    - PHID

- **P.S.** Think you “need” OTG??…

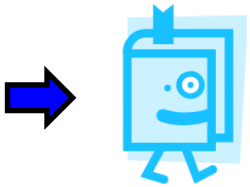


Why LibUSB?

- No class exists for what you want to do
  - Examples
    - HID
      - HID class only has control and interrupt endpoints – you need bulk / isochronous
      - You don't want to learn all about report descriptors

        Don't want to 'waste' code and time parsing them…
    - CDC
      - CDC limited to one bulk in and one bulk out EP – you need more
      - The USBSER.SYS Windows driver causes you grief
- Need to minimize code space
  - LibUSB is free from class protocol overhead
- Host Windows driver does not cooperate / has bug
- You don't want to write a new Windows driver
- To avoid the complexity of a class
- You need to be "Vendor Specific"
  - And again, you don't want to write a new Windows driver
- To quickly try out a complete PC-host / target concept

- **Consider LibUSB!**



USB Basic Concepts

Host - Basics

- Controls enumeration
  - **Reads descriptors** from a USB function to learn about VID/PID, endpoints…
- At lowest level
  - Host initiates transactions - even for interrupt endpoints!
    - Continually polls a device just to see if it has data to transfer
    - Keeps USB protocol "simple" ☺
      (but not very energy efficient)
  - Poll rate specified by the function's EP descriptor
- At application level
  - Host ***or*** function can request to send data!

Function - Basics

- USB "Function" "Device" or "Peripheral"
  ~ the same thing!

  - **Function** = "A USB device that provides a capability to the host, such as an ISDN connection, a digital microphone, or speakers."

  - **Device** = "A logical or physical entity that performs a function"
    =>A device may contain several functions

  - **Peripheral** = Not in spec.
    Could mean the USB hardware of host or function node.

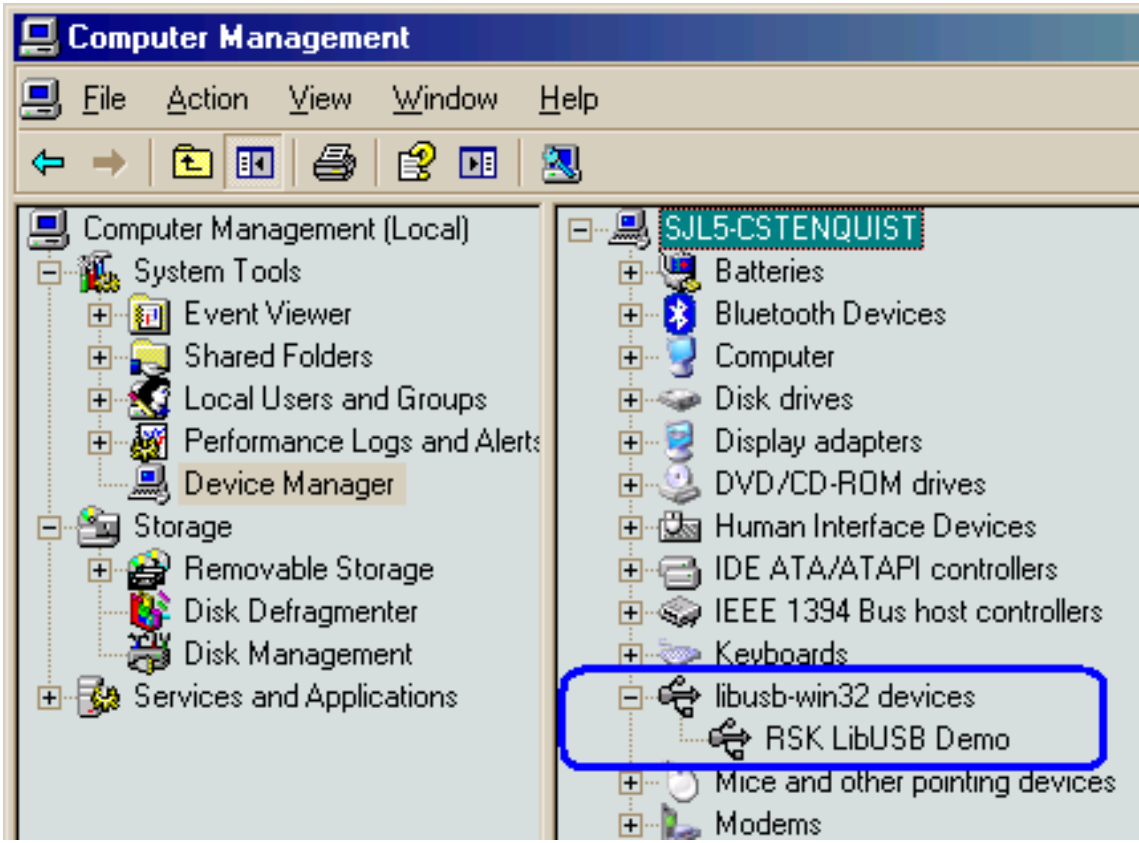
"Peripheral" or "device" <u>could</u> refer to host USB HW – confusing!?

Function - Basics

- Endpoint
  - Have meaning only at application level (host and function)
  - EP number determines meaning of transmitted data (Speed, LED data, display color,..)
  - Defined by function
  - Host learns about the endpoints via the descriptors
    - As you will see in lab
  - Example; “EP1 OUT”
    - “IN” or “OUT” = Dataflow as seen from host

Descriptors - Basics

- Defined by USB function
- At enumeration, host uses **get_descriptor(X)** to learn about function



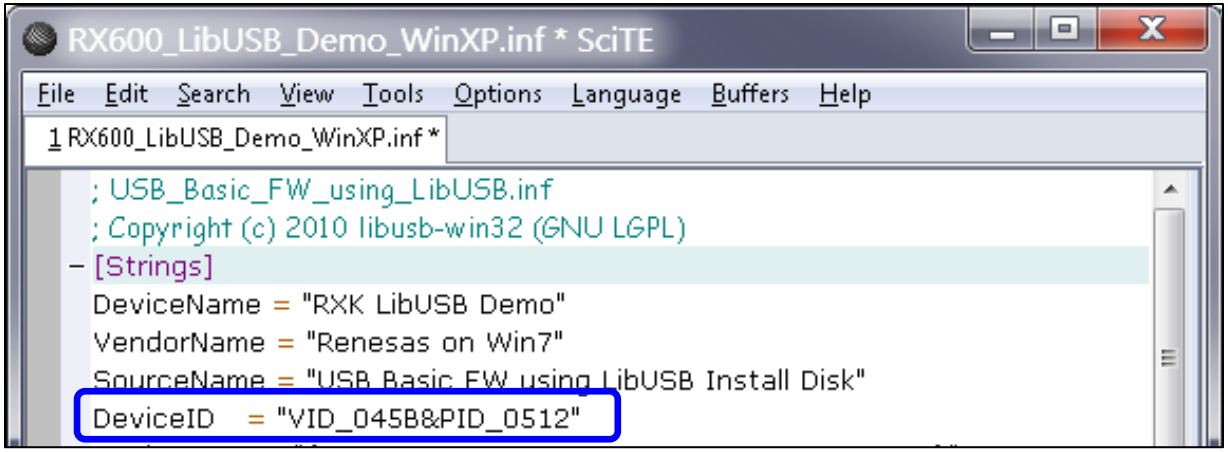
- **Device** descriptor
  - Contains VID & PID
    - PC selects driver listed in INF-file with matching VID-PID-string

```
[DeviceList]
    %USB\VID_8765&PID_1234.DeviceDesc%=USBBulkInstall, USB\VID_8765&PID_1234
```

    - VID from USB Implementers forum
      http://www.usb.org

Enumeration - Basics

- Plug in the device – Host fetches **Device descriptor** containing **VID** and **PID**
- INF file
  - What is its "main" purpose?
    - Used to match **Windows driver** to the VID and PID
    - **DeviceID** = VID + PID string
    - Located in `C:\Windows\system\INF`
      Windows search when new device plugged in
    - If matching INF-file not found?
      - "Found New Hardware" pops up



Enumeration - Basics

RX600_LibUSB_Demo_WinXP.inf * SciTE

File Edit Search ... elp

Example enumeration:

VID=45B, PID=2016 detected…

```
DeviceName = "RXK LibUSB Demo"
VendorName = "Renesas on Win7"
SourceName = "USB Basic FW using LibUSB Install Disk"
DeviceID   = "VID_045B&PID_0512"
```

Windows found matching **DeviceID** in this file.
=> This is the correct INF-file!

```
[SourceDisksFiles.x86]
libusb0.sys     = 1,x86
libusb0_x86.dll = 1,x86

[SourceDisksFiles.amd64]
libusb0.sys     = 1,amd64
libusb0.dll     = 1,amd64
libusb0_x86.dll = 1,x86

[SourceDisksFiles.ia64]
```

What drivers Windows should use

Enumeration - Basics

RX600_LibUSB_Demo_WinXP.inf * SciTE

File Edit Search View Tools ... Help

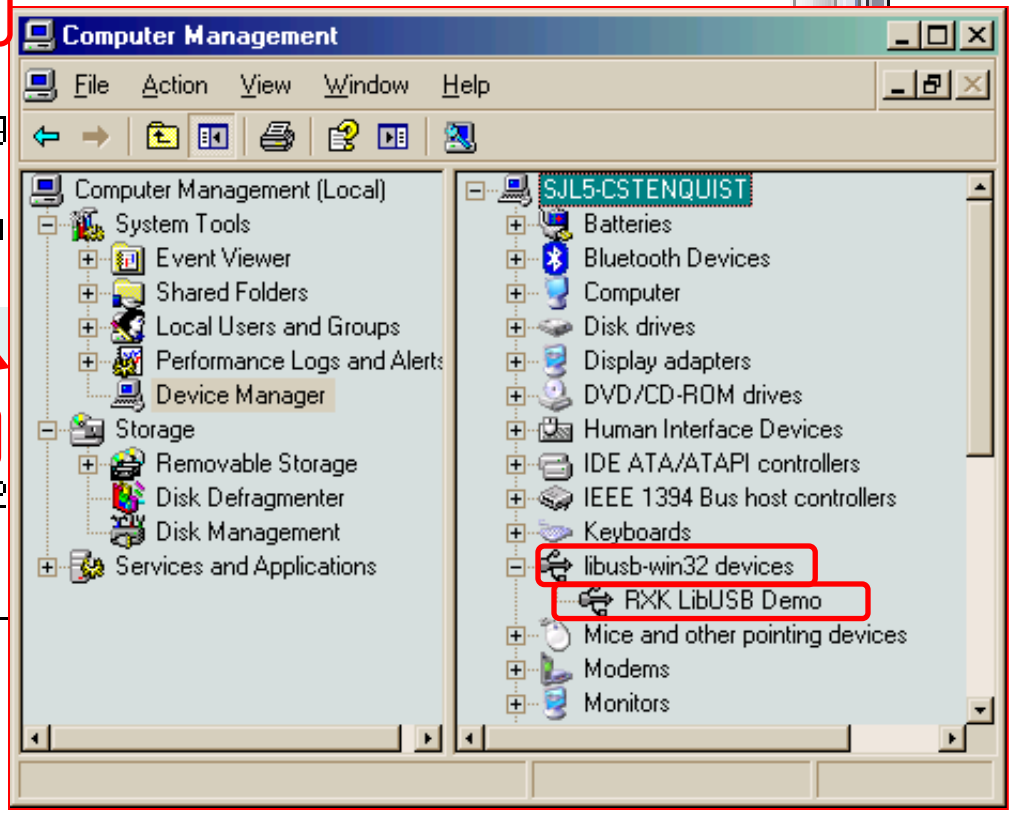
Windows can now use info in INF-file info for multiple purposes

libusb-win32 (GNU LGPL)

```
[Strings]
DeviceName = "RXK LibUSB Demo"
VendorName = "Renesas on Win7"
SourceName = "USB Basic FW using
DeviceID   = "VID_045B&PID_0512"
DeviceGUID = "{326A13F9-414F-40
```

Displayed in **Device Mgr.**

```
[Version]
Signature   = "$Windows NT$"
Class       = "libusb-win32 devices"
ClassGuid   = {EB781AAF-9C70-452
Provider    = "libusb-win32"
CatalogFile = USB_Basic_FW_using_
DriverVer   = 07/24/2011, 1.2.5.0
```



Descriptors - Basics

- **Device** descriptor
  to select host driver

```
uint8_t usb_gplibusb_StringDescriptor2[38] =
{
    38,
    USBC_DT_STRING,
    'R', 0x00,
    'S', 0x00,
    'K', 0x00,
    ' ', 0x00,
    'L', 0x00,
    'i', 0x00,
    'b', 0x00,
    'U', 0x00,
    'S', 0x00,
    'B', 0x00,
    ' ', 0x00,
    'D', 0x00,
    'e', 0x00,
    'm', 0x00,
    'o', 0x00
};
```

- **String** descriptors
  - Describes function with text to host
  - There can be multiple
    - Many different uses

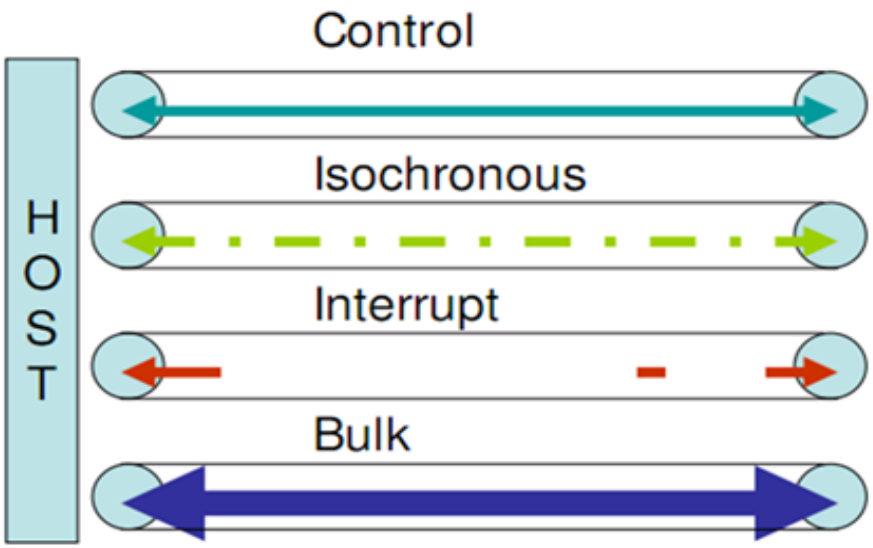
- **Configuration** descriptor
  - Contains
    - **Interface** descriptor(s)
    - **Endpoint** descriptors

```
/* Endpoint 2 Descriptor */
7,                                  /*  0:bLength */
USBC_DT_ENDPOINT,                   /*  1:bDescriptorType */
(uint8_t)(USBC_EP_IN | USBC_EP2),   /*  2:bEndpointAddress */
USBC_EP_BULK,                       /*  3:bmAttribute */
64,                                 /*  4:wMaxPacketSize_lo */
0,                                  /*  5:wMaxPacketSize_hi */
0                                   /*  6:bInterval */
```

Transfer Types - Basics

- Determines <u>how</u> 1 ms (FS) frames are filled with user data
  - Set for each endpoint in “Pipe Information Table” (PIT)

- Control
  - Used for EP0, control commands at e.g. enumeration:
    “Standard Device Requests” = get_descriptor, set_interface,..
    - *(Can* be used by Vendor Specific application for user data)
- Interrupt
  - Max latency guaranteed; 1-255 ms
  - *Example: keyboard, mouse*
- Isochronous
  - Bandwidth guaranteed (latency is not)
  - Fixed nr of bytes per frame
  - *Example: Audio*
- Bulk
  - For time insensitive data
  - 1 ms frame is filled with bulk data after other types have taken what they need
  - *Example: mass storage*



Pipe - Basics

- Node's "channel" of communication for certain endpoint
- Pipe ⇔ endpoint pairing in PIT
  - Function PIT defined statically at compile time
  - Host PIT setup at enumeration

```
/* Pipe information table comprises the following six items (uint16_t × 6).
1. Pipe window select register (address 0x64)
2. Pipe configuration register (address 0x68)
3. Pipe buffer designation register (address 0x6A)
4. Pipe maximum packet size register (address 0x6C)
5. Pipe period control register (address 0x6E)
6. FIFO port usage method */

uint16_t usb_gplibusb_SmplEpTbl1[] =
{
    /*** PIPE1 Definition ****************************************************/
    USBC_PIPE1,
    USBC_BULK | USBC_BFREOFF | USBC_DBLBON | USBC_CNTMDOFF | USBC_SHTNAKON | USBC_DIR_P_OUT | USBC_EP1,
    (uint16_t)USBC_BUF_SIZE(512u) | USBC_BUF_NUMB(8u),
    64,
    0,
    USBC_CUSE,

    /*** PIPE2 Definition ****************************************************/
    USBC_PIPE2,
    USBC_BULK | USBC_BFREOFF | USBC_DBLBON | USBC_CNTMDOFF | USBC_SHTNAKON | USBC_DIR_P_IN | USBC_EP2,
    (uint16_t)USBC_BUF_SIZE(512u) | USBC_BUF_NUMB(24u),
    64,
    0,
    USBC_CUSE,

    /*** PIPE6 Definition ****************************************************/
    USBC_PIPE6,
    USBC_INT | USBC_BFREOFF | USBC_DBLBON | USBC_CNTMDOFF | USBC_SHTNAKON | USBC_DIR_P_IN  | USBC_EP3,
    (uint16_t)USBC_BUF_SIZE(512u) | USBC_BUF_NUMB(40u),
    32u, //16 orig. $REA    Maximum packet size
    0u,
    USBC_CUSE,

    USBC_PDTBLEND,
};
```

Pipe - Basics

- PIT element members
  - Pipe ⇔ Endpoint number
  - Direction of data (IN, OUT)
  - Transfer type
  - etc…

```
/* PIPE2 Definition */
USB_PIPE2,
USB_BULK | USB_BFREOFF | USB_DBLBON | USB_SHTNAKON | USB_DIR_P_IN | USB_EP2,
USB_NONE,
64,
USB_IFISOFF | USB_IITV_TIME(0u),
USB_CUSE,
```

What is a USB Class? - Basics

- Group of similar functionality services
- Predefined descriptor set
- Same host driver (No ‘Found New HW’)
- USB classes defined by USB-IF

  http://www.usb.org/developers/devclass_docs#approved

www.usb.org/developers/devclass_docs#approved

Search

Home | About USB-IF | Channel | Press | Developers | Members | Products

- SuperSpeed USB
- Wireless USB
- Hi-Speed USB
- USB On-The-Go and Embedded Host
- Tools
- USB-IF eStore
- Documents
- USB-IF Compliance Program
- USB FAQ
- Events
- Join USB-IF, Inc.
- Resources

Home > Developers > Approved Device Class Document Download

Approved Class Specification Documents

Audio Class

Audio Device Class Spec for Basic Audio Devices and Adopters Agreement (.zip format, size 1.33MB)
Audio Device Document 1.0
Audio Data Formats 1.0
Audio Terminal Types 1.0
USB MIDI Devices 1.0
Audio Devices Rev. 2.0 Spec and Adopters Agreement (.zip format, size 1.28MB)

Battery Charging

Battery Charging v1.2 Spec and Adopters Agreement (.zip format, size 589 KB)
Battery Charging v1.1 Spec and Adopters Agreement (.zip format, size 292 KB)

Cable and Connector

USB 3.0 Connectors and Cable Assemblies Document Rev. 1.02
Cable and Connector Class 2.0
Series 'A' Plug form factor Guideline 1.0
USB Connector for Mezzanine Applications Guidelines Rev. 1.0
Micro-USB Cables and Connectors v1.01 Spec and Adopters Agreement

Common Class (CCS)

Common Class Base Specification 1.0

Communications Device Class

Class definitions for Communication Devices 1.2 (.zip file format, size 3.42 MB)
the components of CDC 1.1 have been reorganized as five separate documents and associated errata:
ATM120.pdf -- CDC Subclass for Asynchronous Transfer Mode Devices
CDC120-20101103-track.pdf - CDC Subclass for Communications Devices

Vendor Specific - Basics

- Class does allow some descriptor data to be unique
  - vendor string
  - capabilities
  - data format

  …without being *"Vendor Specific"*

- If *any* descriptor item is set to **0xFF**;
  - Not a class! ⇨ "Vendor Specific"
    - New INF-file needed!
    - New host driver needed

LibUSB

What is LibUSB?

An API for host (PC) applications

Includes PC driver to enable PC-application ⟺ USB function communication

- Open source
- Windows XP, Vista, Win7:
  - **Libusb-win32** http://sourceforge.net/apps/trac/libusb-win32/wiki
    - Contains driver LibUSB.dll

      A pre-written, pre-tested, generic driver
- Linux:
  - Even easier for user than for Windows:
    - USB kernel driver **usbfs** automatically associates device to LibUSB driver

Application Level Tools that use LibUSB

www.libusb.org/wiki/libusb-1.0

Open Source Projects using libusb-1.0

- libfprint fingerprint scanning http://cgit.freedesktop.org/libfprint
- Microdia Microdial webcam driver with uses isochronous I/O
- Some Japanese smart card thingy: tsniff and its usage
- coldsync-ppp
- libdc1394-2 (uses isochronous I/O)
- Dimax SUB-20
- libnifalcon
- libftdi-1.0
- SANE - Scanner Access Now Easy
- UsbPicProg - Open source and open hardware USB PIC programmer
- Sigrok - Open source logic analyzer software for various USB logic analyzer hardware
- xpiocards
- usbmuxd USB Multiplex Daemon for iPhone and iPod Touch
- UrJTAG Universal JTAG Library
- Exodriver Open source driver for Labjack data acquisition devices
- btstack A Portable User-Space Bluetooth Stack
- madwimax Linux driver for mobile WiMAX devices
- usbutils USB utilities for Linux, includes the usb.ids file and lsusb
- Yubikey personalization Yubikey personalization cross-platform library and tool
- Openkinnect libfreenect Cross-platfrom drivers and libraries for the Microsoft Xbox Kinect device
- Ettus uhd Cross-platform universal hardware driver for Ettus Research products
- OpenNI Open source OpenNI framework

Bindings for other languages

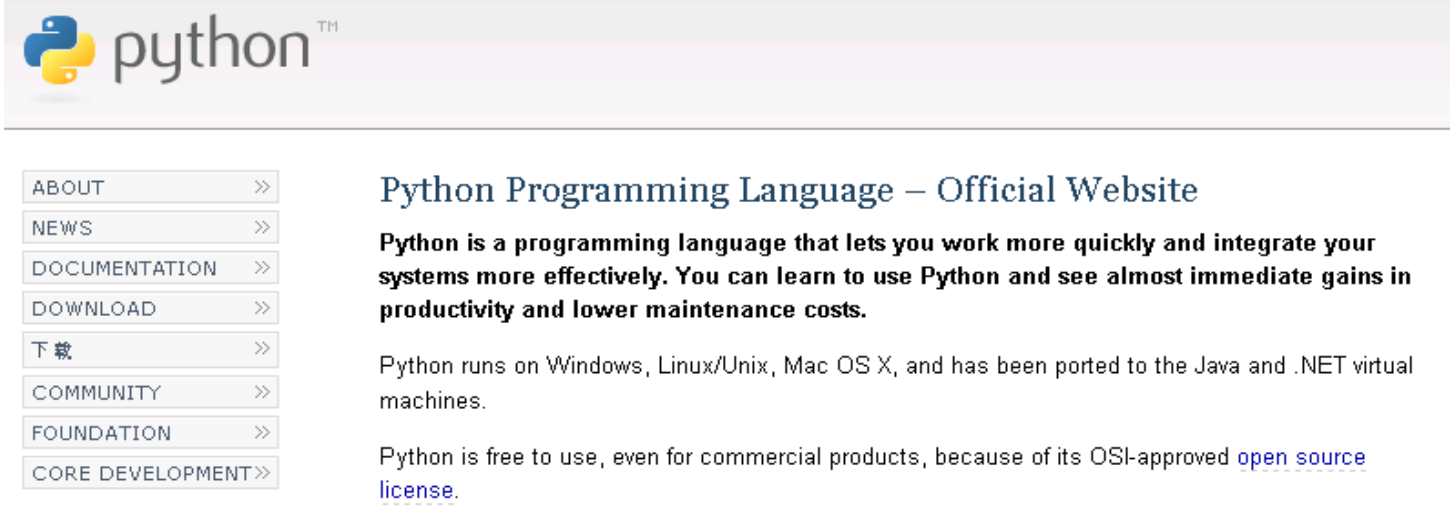
- Haskell bindings: Low level, high level
- RibUSB for Ruby
- pyusb for Python
- libusbdotnet for C#, DotNet and Mono
- partial python wrapper using swig
- python-libusb1, simple Python wrapper
- Go libusb-1.0 wrapper
- node.js libusb-1.0 binding
- OCaml libusb-1.0 binding
- Fator libusb-1.0 binding
- lua libusb-1.0 binding

Lab uses Python & PyUSB

- Python
  - Create PC applications
  - Object-oriented language

```
Object.property = 5          #Set a variable
Object.procedure()           #Run a procedure
```

  - Easy to get started (compared to VC++)?
  - No cost/license
    - www.python.org



  - GUI modules for Python: **Tkinter**, wxPython
- PyUSB
  - Python module from sourceforge.net/apps/trac/pyusb
  - PyUSB accesses the Windows LibUSB driver (LibUSB.dll)

Libusb-win32

- Installs drivers **libusb0.sys**, **libusb0_x86.dll** to `C:\windows\system\drivers`
- INF-file Wizard
- Test tool "Install-filter-win.exe"
  Access ANY device via LibUSB

- The ability to communicate with USB devices referred to as the "**Windows LibUSB Backend**"

LibUSB Calls

- Find USB device(s)
  - find_devices()
- Standard Device requests
  - get_descriptor(x) - Retrieve device descriptor x
  - set_configuration(x) - Set configuration x of device
- R/W to endpoints
  - write_endpoint(), read_endpoint()
- YOU WILL USE ALL THESE IN LAB

Import PyUSB

```
#   C:\Python26\Lib\site-packages\usb..
import usb.core, usb.control, usb.util, time

# Write "help(usb.control)" at the python terminal to ge
#   functions of control.py. Standard Host Device Contro
#   usb.control:
    # 'clear_feature',
    # 'set_feature',
    # 'get_descriptor',
    # 'set_descriptor',
    # 'get_configuration',
    # 'set_configuration',
    # 'get_interface',
    # 'set_interface',
    # 'ENDPOINT_HALT',
    # 'FUNCTION_SUSPEND',
    # 'DEVICE_REMOTE_WAKEUP',
    # 'U1_ENABLE',
```

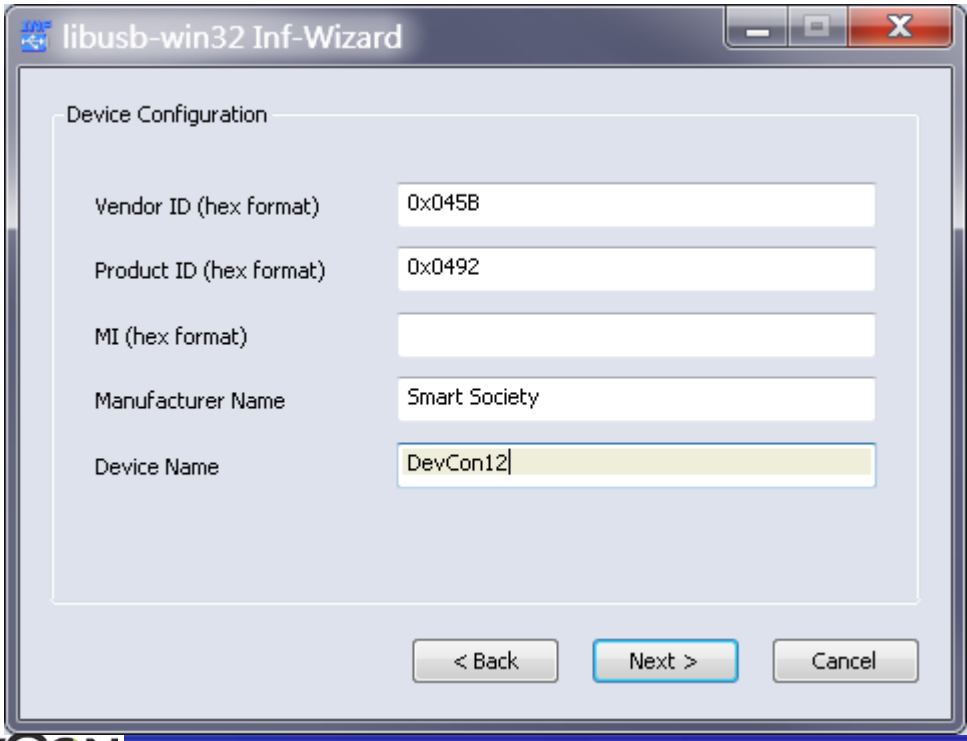
Lab output

```
List USB devices found: [<usb.core.Device object at 0x00C242D0>]
 In detail:
        Vendor ID 0x45b & Product ID = 0x2016
Device with VID 0x45b and PID 0x2016 found:
+++ CONFIGURATION=>INTERFACES=>ENDPOINTS
cfg = <usb.core.Configuration object at 0x00BAE730>
bConfigurationValue = 1:
         bInterfaceNumber = 0, bAlternateSetting = 0
                 bEndpointAddress 0x1
                 bEndpointAddress 0x82
                 bEndpointAddress 0x0

DEVICE DESCRIPTOR: array('B', [18, 1, 0, 2, 255, 255, 255, 64, 91
CONFIGURATION DESCRIPTOR: array('B', [9, 2, 32, 0, 1, 1, 0, 192,
STRING DESCRIPTOR 1 : array('B', [16, 3, 82, 0, 69, 0, 78, 0, 69,
                        *** RENESAS ***
STRING DESCRIPTOR 2 : array('B', [38, 3, 82, 0, 83, 0, 75, 0, 32,
                        *** &RSK LibUSB Demo ***
```

INF-file

- Need to create INF file for your Device ID
  - How to create INF-file?
    - **INF-Wizard**
      - Easy to create INF-file!
      - INF-Wizard starts up by listing all connected devices via LibUSB
      - Any info it cannot get from incoming descriptors it will ask you for
      - Comes with Libusb-Win32



INF-Wizard Result

RX600_LibUSB_Demo_WinXP.inf * SciTE

File Edit Search View Tools Options Language Buffers Help

1 RX600_LibUSB_Demo_WinXP.inf *

```
; USB_Basic_FW_using_LibUSB.inf
; Copyright (c) 2010 libusb-win32 (GNU LGPL)
[Strings]
DeviceName = "RXK LibUSB Demo"
VendorName = "Renesas on Win7"
SourceName = "USB Basic FW using LibUSB Install Disk"
DeviceID   = "VID_045B&PID_0512"
DeviceGUID = "{326A13F9-414F-4078-92
[Version]
Signature   = "$Windows NT$"
Class       = "libusb-win32 devices"
B-A5DF-642A87ECA567}
LibUSB.cat
```

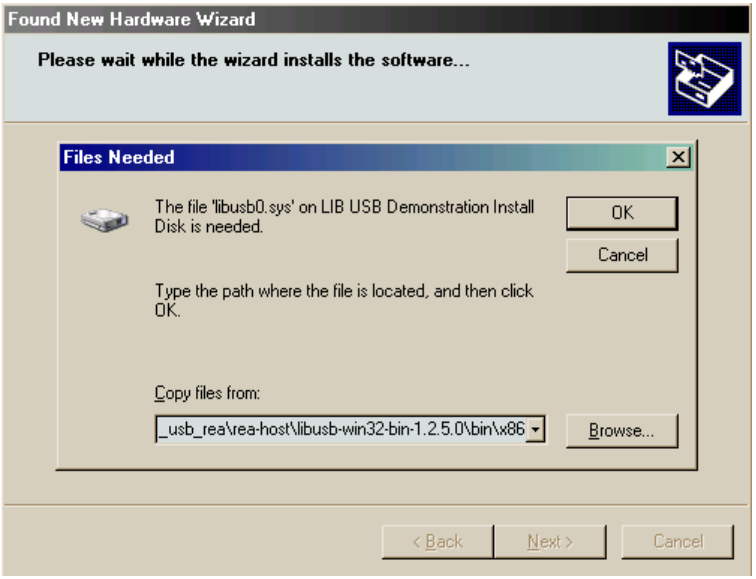
You entered **Vendor** name into INF-Wizard…

Wizard got **DeviceID** (VID, PID) and **Class** (LibUSB) by using LibUSB's *get_descriptor(DEVICE)*

The information in the INF-file comes from different places. The INF-Wizard figured it all out for you!

Customer Install

- Your product install will need to provide the following (Or have the user download):
  - **INF-file** with your DeviceID (VID&PID string)
  - **LibUSB drivers** (libusb0.sys & libusb_x86.dll)
  - A PC user **application**
    - Python, PyUSB?
    - In the Renesas LibUSB download, Python/PyUSB not needed. PyInstaller was used to convert the application to an EXE-file



Lab

- **Host**: “Renesas_libusb_host.py”

- **Function**: LibUSB project **R01AN0492** for RSK62N, or YRDK63N

- The demo
  - Set an LED (host->RX)
  - Write to LCD (host->RX)
  - Take an ADC measurement (host->RX)
    + send it over USB to host (RX->host)

(prototype board)

Lab Endpoints

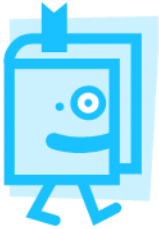
- **Control** (enumeration)
  - EP0 OUT

- **Bulk** (general data transfer)
  - Lab commands to RSK; EP1 OUT
    - Set an LED
    - Write to the LCD
    - Take an ADC measurement
  - Read ADC data from RSK; EP2 IN

- **Interrupt** (e.g. user input)
  - Read demo data when SW1 is pushed; EP3 IN

(prototype board)

Access ANY USB Device! Updates



- You can talk to **ANY USB device** with LibUSB!
  - Learn about connected devices, test, …
  - Run "**LibUSB Install-filter**" - comes with Libusb-win32
    - You enable a device from a list
    - Now you can read descriptors, read/write endpoints *without an associated INF-file*
    - But, *do not change interface or configuration* if device already enumerated by Windows (Linux)
      PC's USB subsystem may crash!!!

LibUSB updates

- **libusbx**
  - Has newer LibUSB library and Windows driver
  - Fully backwards compatible
    - Main site: http://sourceforge.net/apps/mediawiki/libusbx/index.php?title=Main_Page
    - Reasons for: http://libusb.6.n5.nabble.com/libusb-is-dead-long-live-libusbx-td5651413.html

Libusb-Win32 FAQ

sourceforge.net https://sourceforge.net/apps/trac/libusb-win32/wiki/libusbwin32_faq

libusb-win32

Frequently Asked Questions

Can I use libusb-win32 to open a file on a USB storage device?

Yes in theory, libusb can be used for low-level communication with USB Mass Storage Class devices
the particular filesystem used on the device, most commonly FAT32. So this is not a simple task. Th
provide file API for the task.

Can I use libusb-win32 to talk to a device where the vendor driver is not good enough?

Yes, libusb-win32 can be used for low-level communication with USB devices. However, in order to
the device works. So libusb-win32 only provides the possibilities to do this and you will have to do t

You can try to contact the vendor to see if they can provide you the communication protocol, or y
existing libusb based application under Linux (or BSD or Mac OS X), that can be used as a good refe

Can I use libusb-win32 on HID class device?

Yes you can. You can use the filter driver with the HID device and then you can use libusb-win32 t

Questions?

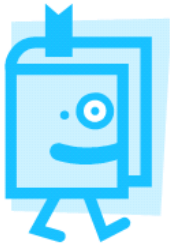
Connect to the Smart Society with USB!

Smart Society Challenge

- You need to
  - get your idea to market SOON
  - avoid the complexity of a class
  - quickly try out a complete PC-host / target solution
  - be “Vendor Specific” but don’t want to write a new Windows driver
- You don’t want to learn all about HID descriptors etc..

Solution

- ***Use LibUSB!***
- **Do you agree?**

RX USB UNIVERSAL SERIAL BUS **+**  **=** 

Summary

- Use LibUSB when
  - No class exists for what you want to do
    - Examples
      - HID
        - HID class only has control and interrupt endpoints – you need bulk / isochronous
        - You don't want to learn all about report descriptors

          Don't want to 'waste' code and time parsing them…
      - CDC
        - CDC limited to one bulk in and one bulk out EP – you need more
        - The USBSER.SYS Windows driver causes you grief
  - You need to minimize code space
    - LibUSB is free from class protocol overhead
  - The host Windows driver does not cooperate / has bug
  - You don't want to write a new Windows driver
  - You want to avoid the complexity of a class
  - You need to be "Vendor Specific"
    - And again, you don't want to write a new Windows driver
  - You need to quickly try out a complete PC-host / target concept

DevCon

Enabling the Smart Society

Renesas

Renesas Electronics America Inc.