WINE and HID

HID Devices and WINE

Presented 2015 WineConf
Vienna, Austria

By Aric Stewart

Presentation Goals

I would like to get you all generally familiar with the HID architecture and what I am trying to implement and why I am taking the approach I have chosen

Someone here may actually be interested enough to want to help and this gives some good foundations.

Why work on this?

- Platform specific gamepad (joystick) code duplicated in a variety of locations, bring them all together to a common place
  - dinput, winmm
  - xinput(future), rawinput(future)
- Improve performance and functionality
  - mice in dinput
- Some applications work directly with HID devices and hid.dll

What this is not

- Will not magically fix all joysticks
- Will not magically make the xbox 360 controller work
- Will not magically make xinput work
- Will (mostly) not be directly user visible at all

Outline

- Hid Architecture
- Plug and Play
- Details on HID internals
  - Platform Specific Details and Issues
  - OS X
  - Linux
- Ask Question at any time

Architecture Starting Points

- I am not a windows kernel developer
  - But that is ok, this is not windows kernel development
- The proposed HID architecture is 'Driver Like'
  - But we are not really a driver
- Native HID device drivers? Not likely...
  - Why? It would be really neat...

HID User architecture

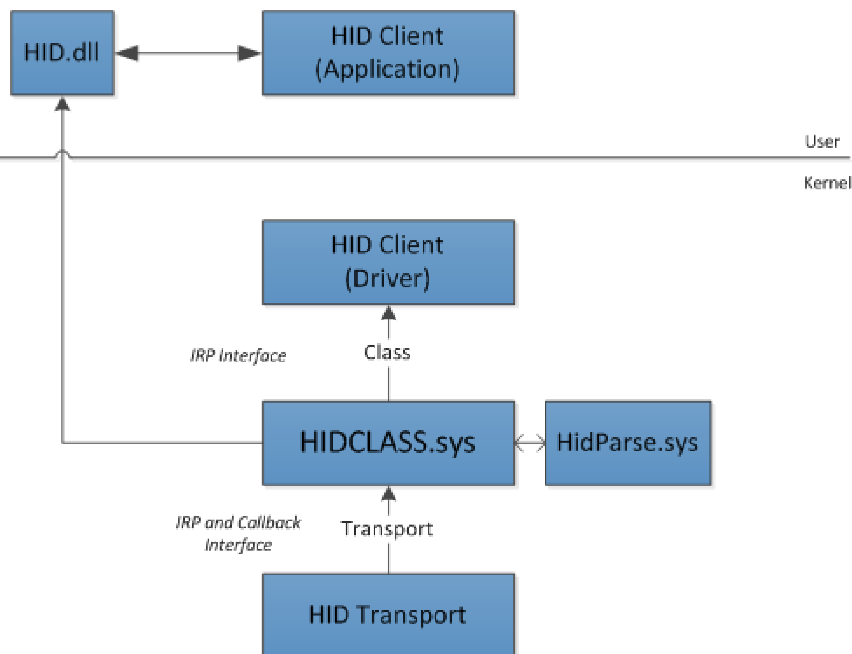


Image courtesy of MSDN

User Space programs can use HID.dll in 2 ways.

- Call hid.dll APIs HidD_xxx and HidP_xxx functions
- Directly send HID class IOCTLS to the device

Either way the HID device is a system device that is detectable via SetupAPI and is expected to be able to be Opened, Closed, Read and Written to.

HID 'Kernel' Stack

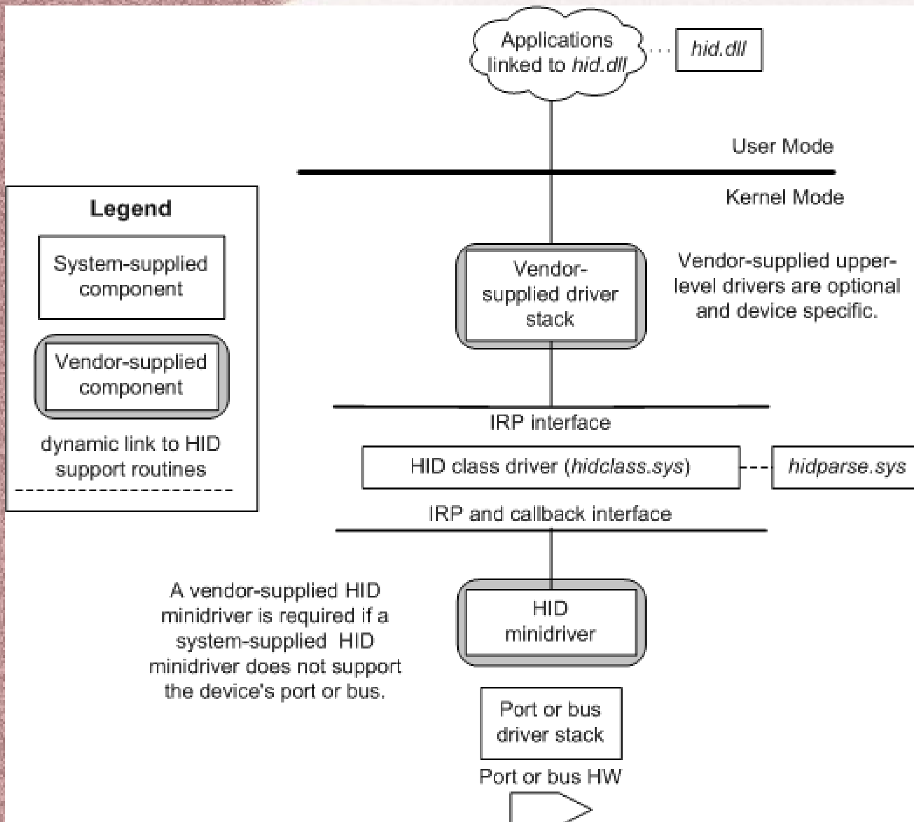


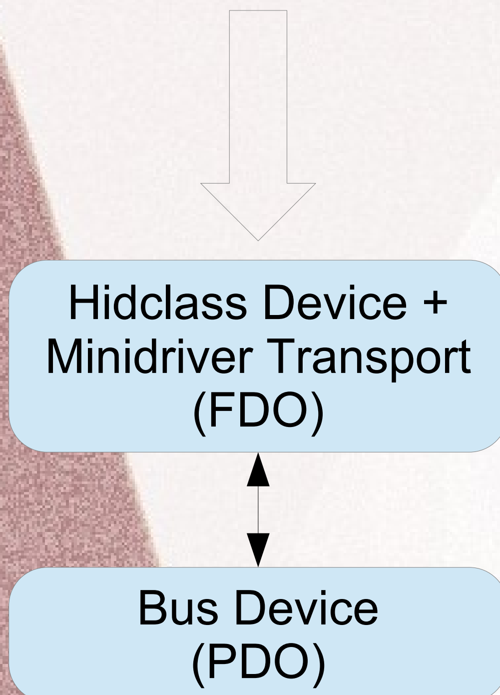
Image courtesy of MSDN

The HID device that the user mode applications or dlls is accessing is actually a class driver, or driver pair, which represents the underlying communications directly to the hardware.

Device Stacks

A 'device' is not simply a hardware thing plugged into the system. A 'device' is an object , accessible via CreateFile that response to various IOCTLs and hardware style API calls. Most hardwares devices actually have long chains of these objects called Device Stacks.

Don't be scared by the big MSDN Stacks:
Really all we care about is this:



FDO : Function Device, Generally describes the device that talks UP the driver stack.

PDO: Physical Device, Generally describes the device that talks DOWN the driver stack.

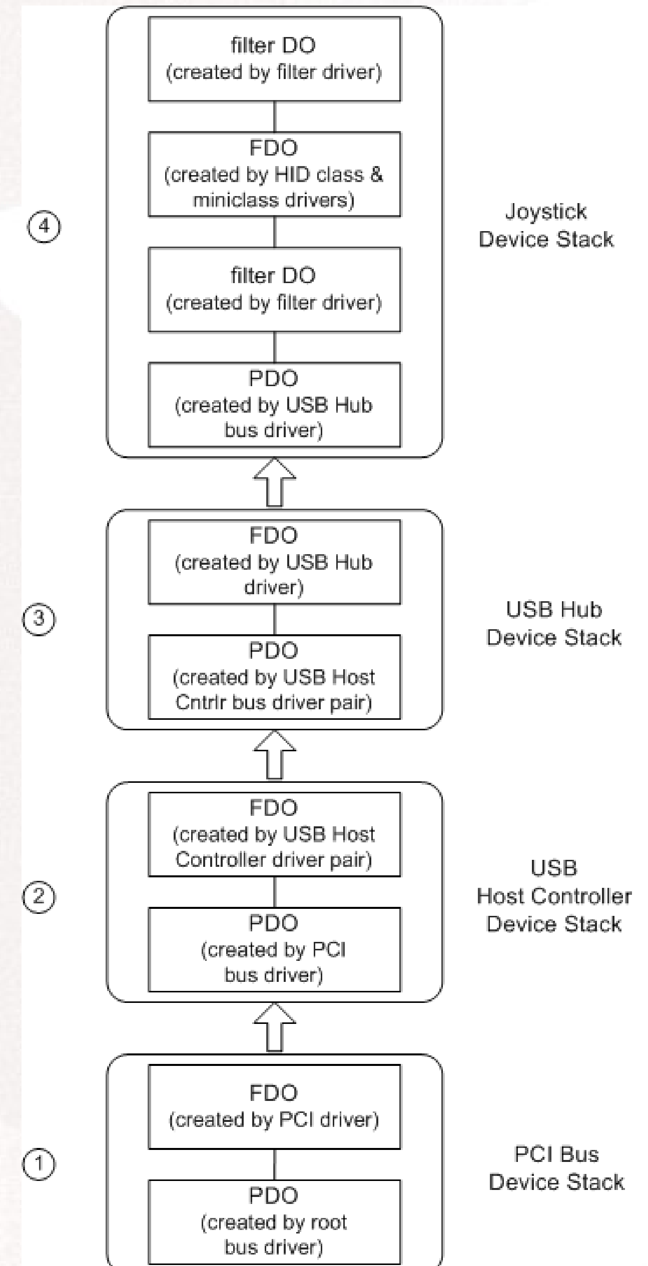


Image courtesy of MSDN

Class Driver / Minidriver

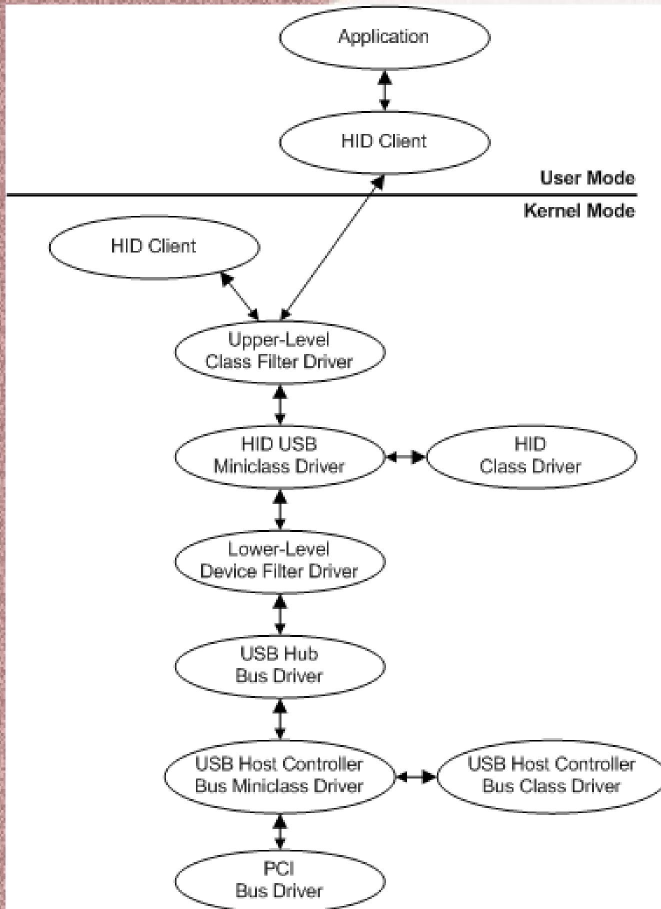


Image courtesy of MSDN

Two sides of the same coin

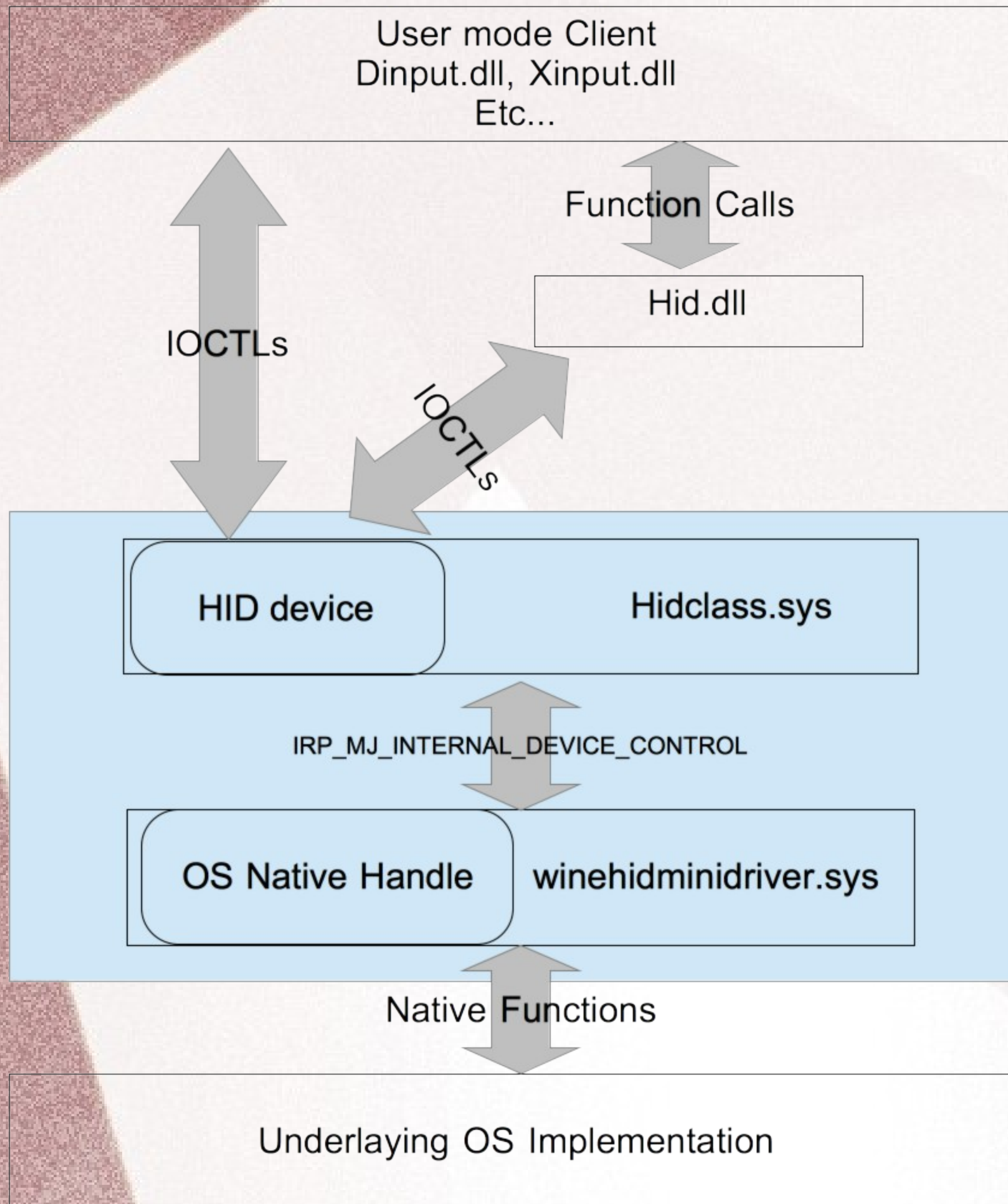
- Class Drivers provide most of the upper level facing interfaces

- Minidrivers are the transport drivers
- The minidriver access the physical device and lower drivers in the chain

Minidriver calls a registration function in the class driver on DriverEntry. The class driver takes over many of the driver entry points, such as handling IOCTLs, Reads, Writes and AddDevice

Together they are the top level FDO for the HID device's device stack.

WINE's HID architecture



The area in blue is our pseudo-driver (FDO) area.

- We need the client facing parts to be as exact to windows as we can. Devices, ioctls, etc...
- Our hidclass.sys and minidrivers are living in user mode
- Get to ignore 90% of the complexities of windows driver development
- Only 1 device, don't really have a bus device or a device stack as our PDO is generally just a handle to the native device
- Platform specific code all goes in the minidriver winehidminidriver.sys

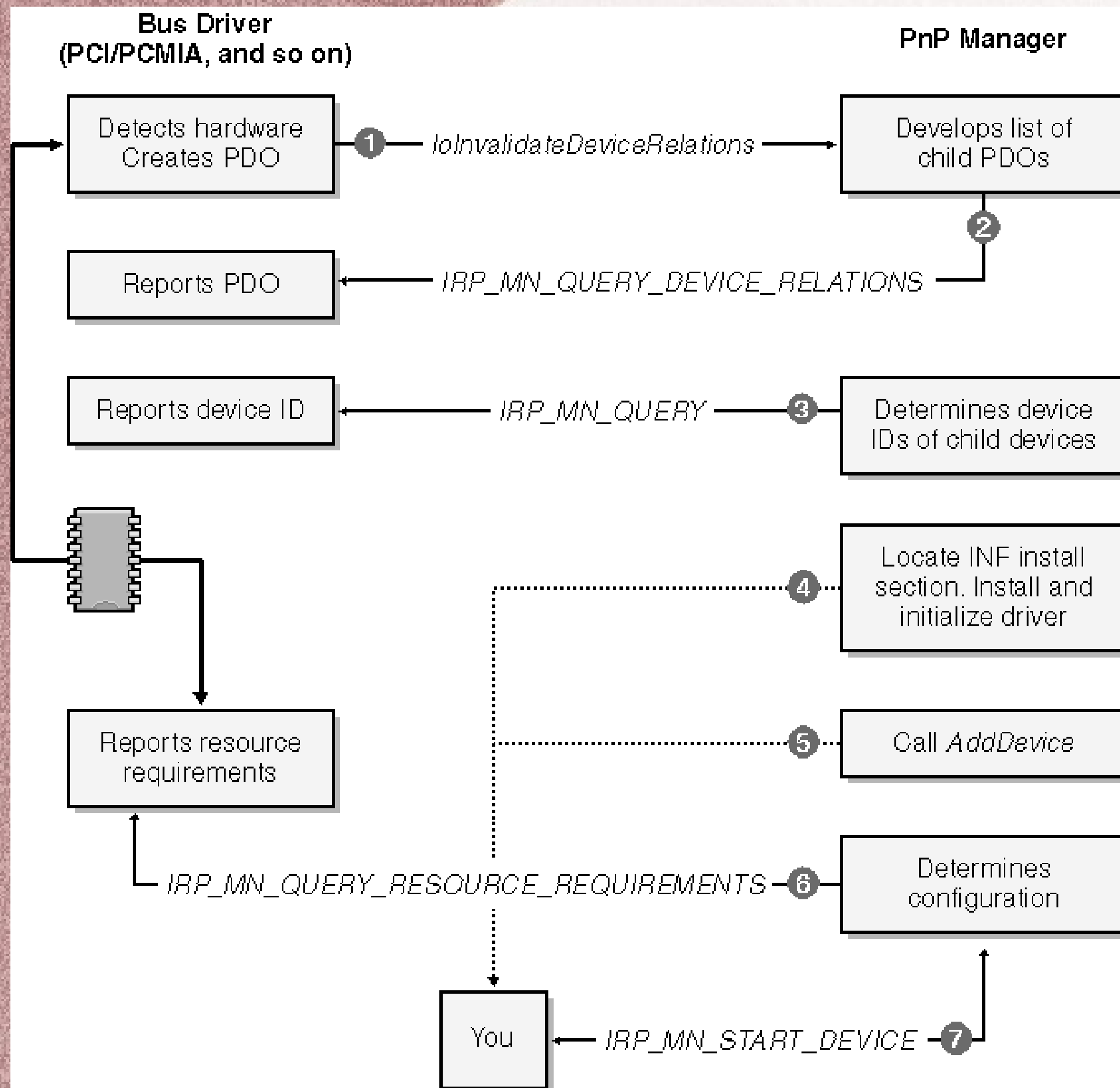
Why no Windows Native Minidrivers?

- They expect kernel functions to exists
  - Most of which are stubs at best
- They expect to communicate to, at best, a bus device, or even directly to hardware
- Require functional Plug and Play driver detection loading and device creations processes
- Cannot think of any seriously useful examples or demand

Plug and Play
(The parts we care about)

- Bus Enumeration of devices
  - On power-up do the insertion process for every located device
- Driver locating and loading
- Hot plug insertion and removal of devices
- Notifications up to the user level
  - RegisterDeviceNotification

PnP on Windows



- The bus device is responsible for enumerating devices.
- When a new device is found a message is sent to the PnP Manager which locates a driver, loads and initializes the driver, if not already loaded
- Then the PnP Manager calls AddDevice in the driver to setup the device with the bus PDO.
- Finally IRP_MN_START_DEVICE is sent

Image courtesy of “Programming the Microsoft Windows Driver Model”
By Walter Oney

PnP Insertion / Removal

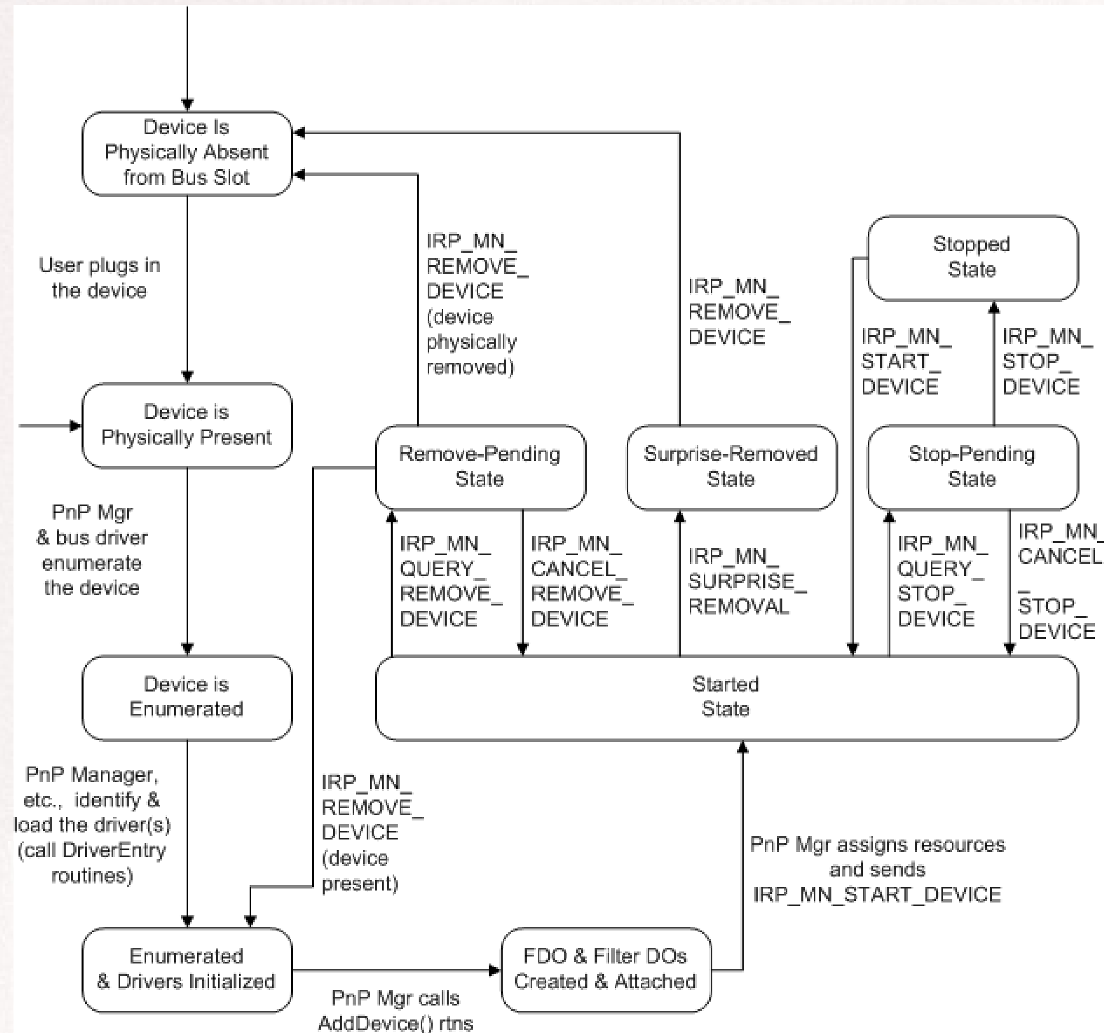


Image courtesy of MSDN

PnP in WINE

- We don't have anything really
  - No bus drivers
  - PlugPlay service is a stub
- Rough plug and play enumeration and discovery implemented in hidclass and the HID minidriver not cleanly separated
- Do we need more?
  - It would be neat but really what do we need?

HID / hidclass Internals

- Feel free to check out now if your interest has been satisfied
- MSDN has a lot of good information
- Generally driver development books are less helpful
- “Programming the Microsoft Windows Driver Model” By Walter Oney has a great chapter (http://flylib.com/books/en/4.168.1.84/1/)

Technical Details

- The USB HID specification
- Devices and SetupAPI
- Partner Drivers / Minidrivers
- Wine details
- Platform details
- HID Clients

HID and USB

- HID devices are not required to be USB
- However the HID specification is tightly coupled with the USB specification
- Non-USB devices fake USB information
- Constant values such as Usages and Usage Pages all match the USB spec

USB and HID

- Given a device
  - get device information (VendorID, ProductID, Strings, etc...)
  - get the report descriptor
  - read and write reports to the device
- Turn HID Report Descriptors into PHIDP_PREPARSED_DATA for the HidP_XxX functions
- Read and write individual data elements in a report

Report Descriptors

- Input Reports, Output Reports and Feature Reports

```
0x05, 0x01,          // USAGE_PAGE (Generic Desktop)
0x09, 0x05,          // USAGE (Game Pad)
0xa1, 0x01,          // COLLECTION (Application)
0xa1, 0x00,          //   COLLECTION (Physical)
0x85, 0x01,          //     REPORT_ID (1)
0x05, 0x09,          //     USAGE_PAGE (Button)
0x19, 0x01,          //     USAGE_MINIMUM (Button 1)
0x29, 0x10,          //     USAGE_MAXIMUM (Button 16)
0x15, 0x00,          //     LOGICAL_MINIMUM (0)
0x25, 0x01,          //     LOGICAL_MAXIMUM (1)
0x95, 0x10,          //     REPORT_COUNT (16)
0x75, 0x01,          //     REPORT_SIZE (1)
0x81, 0x02,          //     INPUT (Data,Var,Abs)
0x05, 0x01,          //     USAGE_PAGE (Generic Desktop)
0x09, 0x30,          //     USAGE (X)
0x09, 0x31,          //     USAGE (Y)
0x09, 0x32,          //     USAGE (Z)
0x09, 0x33,          //     USAGE (Rx)
0x15, 0x81,          //     LOGICAL_MINIMUM (-127)
0x25, 0x7f,          //     LOGICAL_MAXIMUM (127)
0x75, 0x08,          //     REPORT_SIZE (8)
0x95, 0x04,          //     REPORT_COUNT (4)
0x81, 0x02,          //     INPUT (Data,Var,Abs)
0xc0,                //   END_COLLECTION
0xc0                 // END_COLLECTION
```

- Used to generate the opaque PHIDP_PREPARSED_DATA structure used on the HidP_XXX functions around processing reports.
- HidD_GetPhysicalDescriptor (IOCTL_GET_PHYSICAL_DESCRIPTOR)

HID Reports

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Report ID (1)							
Byte 1	Button 8	Button 7	Button 6	Button 5	Button 4	Button 3	Button 2	Button 1
Byte 2	Button 16	Button 15	Button 14	Button 13	Button 12	Button 11	Button 10	Button 9
Byte 3	Left Stick X Axis as Signed Char Integer (X)							
Byte 4	Left Stick Y Axis as Signed Char Integer (Y)							
Byte 5	Right Stick X Axis as Signed Char Integer (Z)							
Byte 6	Right Stick Y Axis as Signed Char Integer (Rx)							

- Input reports acquired by user client through ReadFile operations
- Input reports acquired by the hidclass from the minidriver using IOCTL_HID_READ_REPORT or IOCTL_HID_GET_INPUT REPORT
- Output report and Function reports (leds, FF, etc..) sent by the user client through WriteFile

- To the user, reports are opaque data, individual controls are access using the hid.dll APIs such as HidP_GetUsageValue, HidP_GetButtons, HidP_SetUsages, etc...

HID devices and SetupAPI

\?\HID#vid_vvvv&pid_pppp&mi_ii#aaaaaaaaaaaaaaaa#{4d1e55b2-f16f-11cf-88cb-001111000030}

- &mi_ii or &ig_ii (Xinput compatible)
- SetupDiGetClassDevs and friends
- CreateFile
- What are 'Top Level Collections'?
  - A multifunction device (mouse/keyboard) may have more than 1 top level collection
  - Ends up appearing as 2 separate HID devices

The minidriver connection

- The hidclass.sys class driver is a library present to make writing HID drivers much simpler
- In DriverEntry calls the registration function it replaces a lot of entry points
- Uses Internal IOCTLS to communicate

Windows Builtin Minidrivers

There are not a lot of minidrivers out there because Windows provides quite a number of built-in mindrivers to cover most of the existing common bus types on a system.

USB	Hidusb.sys	Win 2000+
Bluetooth	Hidbth.sys	Win Vista+
Bluetooth LE	Hidbthle.sys	Win 8+
I2C	Hidi2c.sys	Win 8+
General Purpose IO	Hidinterrupt.sys	Win 10+
GamePort	Hidgame.sys	Win Xp – WinVista

In Wine

- Just starting to get into Wine. Basic structure approved by Alexandre
- Winehidminidriver.sys loaded all the time at initialization as a system service
- Loads the appropriate bits for the platform transport and handles all the PnP enumeration and insertion logic
- Hidclass can support multiple mindriver registrations in 1 instance
- Try to mostly support/preserve the HidClass / Minidriver separation

In Wine...

- Theoretically other minidrivers could be written.
- All platform specific code lives here
- PHIDP_PREPARSED_DATA generation [proposed]
  - IOCTL_WINE_HID_GET_PREPARSED_SIZE
  - IOCTL_WINE_HID_GET_PREPARSED

OS X

- IOHIDxxx APIs give us just what we want
- Direct access to ReportDescriptors and Reports from the device
- Include access to keyboard and Mouse devices.
- Mostly complete for Input Reports
- Output reports presently unimplemented
- Force Feedback is missing from descriptors and will have to be built by hand

Linux, hidraw

- My Linux knowledge is much more limited...
- hidraw gives us direct access to everything we want, Descriptors, reports, read, write...
- It is considered internal and none of the devices are given user access
- Xbox devices, and maybe others, do not appear in hidraw

Linux, input

- No access to reports or descriptors, would have to build all the connections by hand
- No access to most of the underlying device information; usb strings, serial numbers, usages for device or elements
- Have to hand build Report Descriptors and Reports

Hid Clients

- Raw Input
- Dinput / Xinput/ Winmm
- Of course, direct HID access from Applications

RawInput

- Mostly lightly wrapped HID
- WM_INPUT messages
- RegisterRawInputDevices
- Need device discovery, access to the messaging system

Dinput, Xinput, Winmm

- Dinput reported as the default windows client for Joystick (0x1 0x4) and GamePad devices (0x1 0x5) devices
- Having them all as HID clients eliminates duplicated platform specific code
- Hopefully clears the path for cleaner implementations for xinput.dll

Want to help?

- Thanks for all the code review!
- Linux input developer
  - The linux minidriver?
- The HID clients, RawInput, xinput etc...
- Comment? Questions? Suggestions?
- Thanks!