User-Level Device Access

Moving beyond KEXTs

Session 207

Dean Reece
I/O Kit Team Manager

These are confidential sessions—please refrain from streaming, blogging, or taking pictures

Introduction

Only on Mac OS

Dean Reece

- Techniques for user-space access to devices
- Summary of available APIs

Thane Norton

- User-space HID device driver walkthrough

Ethan Bold

- Sleep/wake from user-space
- New sleep behaviors

What You Will Learn

- The Mac App Store and drivers
- Benefits and challenges of user-level device software
- Basics of device matching
- Running a process when a device is attached
- Finding and accessing devices from an app
- A tour of user-level device APIs

Drivers and the Mac App Store

- Guidelines for Mac App Store restrict many driverlike behaviors
  - No kernel extensions (KEXTs)
  - No privileged execution
  - App must be self-contained within its bundle
  - Background execution requires user approval

Drivers and the Mac App Store

- Access to many restricted APIs are allowed on as-needed basis
  - Mac App Store applications are run inside of an "App SandBox"
  - "Entitlements" allows access to restricted APIs
  - `com.apple.security.device.microphone` for microphone access

App Sandbox and the Mac App Store	Nob Hill Tuesday 3:15PM

User-Space Device Software Advantages

- Simpler development and debugging
  - Shares APIs and frameworks with other apps
  - Easier debugging than KEXTs
  - Richer performance tools available
- Improved customer experience
  - Faults will not crash the system
  - May not require admin credentials to install/use
  - Does not prevent distribution via Mac App Store

User-Space Device Software Challenges

- Limited device access
  - Not all device types are directly accessible to user-space
  - Only a few device types are publishable from user-space
- Several different APIs for device access
  - I/O Kit, Bluetooth, POSIX, CUPS, etc.
  - No “stacking” model like IORegistry
- Performance
  - Increased scheduling latency (batching)
  - Subject to paging stalls

The IORegistry

- Collection of active devices
  - Kernel-resident
  - Apps can browse IORegistry

```
Built-in iSight@fd11  <class IOUSBDevice, …>
{
  IOCFPlugInTypes = { … }
  IOUserClientClass = IOUSBDeviceUserClientV2
  PortNum = 1
  USB Address = 3
  USB Product Name = "Built-in iSight"
  USB Serial Number = "8TA5E1EGXDDQ4L00"
  USB Vendor Name = "Apple Inc."
  bDeviceClass = 239
  bDeviceProtocol = 1
  bDeviceSubClass = 2
  bMaxPacketSize0 = 64
  bNumConfigurations = 1
  bcdDevice = 582
  idProduct = 34050
  idVendor = 1452
  locationID = 18446744073660334080
}
```

The IORegistry

- Collection of active devices
  - Kernel-resident
  - Apps can browse IORegistry
- Each device is an object
  - Class indicates capabilities

```
Built-in iSight@fd11  <class IOUSBDevice, …>
{
  IOCFPlugInTypes = { … }
  IOUserClientClass = IOUSBDeviceUserClientV2
  PortNum = 1
  USB Address = 3
  USB Product Name = "Built-in iSight"
  USB Serial Number = "8TA5E1EGXDDQ4L00"
  USB Vendor Name = "Apple Inc."
  bDeviceClass = 239
  bDeviceProtocol = 1
  bDeviceSubClass = 2
  bMaxPacketSize0 = 64
  bNumConfigurations = 1
  bcdDevice = 582
  idProduct = 34050
  idVendor = 1452
  locationID = 18446744073660334080
}
```

The IORegistry

- Collection of active devices
  - Kernel-resident
  - Apps can browse IORegistry
- Each device is an object
  - Class indicates capabilities
  - Each object has properties
    - To match drivers to devices
    - Device Vendor and Model ID

```
Built-in iSight@fd11  <class IOUSBDevice, …>
{
  IOCFPlugInTypes = { … }
  IOUserClientClass = IOUSBDeviceUserClientV2
  PortNum = 1
  USB Address = 3
  USB Product Name = "Built-in iSight"
  USB Serial Number = "8TA5E1EGXDDQ4L00"
  USB Vendor Name = "Apple Inc."
  bDeviceClass = 239
  bDeviceProtocol = 1
  bDeviceSubClass = 2
  bMaxPacketSize0 = 64
  bNumConfigurations = 1
  bcdDevice = 582
  idProduct = 34050
  idVendor = 1452
  locationID = 18446744073660334080
}
```

The IORegistry

- Collection of active devices
  - Kernel-resident
  - Apps can browse IORegistry
- Each device is an object
  - Class indicates capabilities
  - Each object has properties
    - To match drivers to devices
    - Device Vendor and Model ID

Run `ioreg -l`
or `IORegistryExplorer.app`

```
Built-in iSight@fd11  <class IOUSBDevice, …>
{
  IOCFPlugInTypes = { … }
  IOUserClientClass = IOUSBDeviceUserClientV2
  PortNum = 1
  USB Address = 3
  USB Product Name = "Built-in iSight"
  USB Serial Number = "8TA5E1EGXDDQ4L00"
  USB Vendor Name = "Apple Inc."
  bDeviceClass = 239
  bDeviceProtocol = 1
  bDeviceSubClass = 2
  bMaxPacketSize0 = 64
  bNumConfigurations = 1
  bcdDevice = 582
  idProduct = 34050
  idVendor = 1452
  locationID = 18446744073660334080
}
```

Matching Dictionary

- Data structure used to describe devices of interest

```
{
  IOProviderClass = IOUSBDevice
  idProduct = 34050
  idVendor = 1452
}
```

```
Built-in iSight@fd11  <class IOUSBDevice, …>
{
  IOCFPlugInTypes = { … }
  IOUserClientClass = IOUSBDeviceUserClientV2
  PortNum = 1
  USB Address = 3
  USB Product Name = "Built-in iSight"
  USB Serial Number = "8TA5E1EGXDDQ4L00"
  USB Vendor Name = "Apple Inc."
  bDeviceClass = 239
  bDeviceProtocol = 1
  bDeviceSubClass = 2
  bMaxPacketSize0 = 64
  bNumConfigurations = 1
  bcdDevice = 582
  idProduct = 34050
  idVendor = 1452
  locationID = 18446744073660334080
}
```

Matching Dictionary

- Data structure used to describe devices of interest
- Details are class specific

```
{
  IOProviderClass = IOUSBDevice
  idProduct = 34050
  idVendor = 1452
}
```

```
Built-in iSight@fd11  <class IOUSBDevice, …>
{
  IOCFPlugInTypes = { … }
  IOUserClientClass = IOUSBDeviceUserClientV2
  PortNum = 1
  USB Address = 3
  USB Product Name = "Built-in iSight"
  USB Serial Number = "8TA5E1EGXDDQ4L00"
  USB Vendor Name = "Apple Inc."
  bDeviceClass = 239
  bDeviceProtocol = 1
  bDeviceSubClass = 2
  bMaxPacketSize0 = 64
  bNumConfigurations = 1
  bcdDevice = 582
  idProduct = 34050
  idVendor = 1452
  locationID = 18446744073660334080
}
```

Matching Dictionary

- Data structure used to describe devices of interest
- Details are class-specific
- USB device match shown here

```
{
  IOProviderClass = IOUSBDevice
  idProduct = 34050
  idVendor = 1452
}
```

```
Built-in iSight@fd11  <class IOUSBDevice, …>
{
  IOCFPlugInTypes = { … }
  IOUserClientClass = IOUSBDeviceUserClientV2
  PortNum = 1
  USB Address = 3
  USB Product Name = "Built-in iSight"
  USB Serial Number = "8TA5E1EGXDDQ4L00"
  USB Vendor Name = "Apple Inc."
  bDeviceClass = 239
  bDeviceProtocol = 1
  bDeviceSubClass = 2
  bMaxPacketSize0 = 64
  bNumConfigurations = 1
  bcdDevice = 582
  idProduct = 34050
  idVendor = 1452
  locationID = 18446744073660334080
}
```

Launch-on-Hardware

- launchd runs your job when matching device appears

```
EnableTransactions = 1
Label = "com.mycompany.xyzdevice_attached"
LaunchEvents = {
    "com.apple.iokit.matching" = {
        "xyzdevice-attached" = {
            IOProviderClass = IOUSBDevice
            idProduct = 34050
            idVendor = 1452
        }
    }
}
ProgramArguments = ( "/Applications/MyApp.app/Contents/MacOS/MyApp" )
```

Launch-on-Hardware

- launchd runs your job when matching device appears

```
EnableTransactions = 1
Label = "com.mycompany.xyzdevice_attached"
LaunchEvents = {
    "com.apple.iokit.matching" = {
        "xyzdevice-attached" = {
            IOProviderClass = IOUSBDevice
            idProduct = 34050
            idVendor = 1452
        }
    }
}
ProgramArguments = ( "/Applications/MyApp.app/Contents/MacOS/MyApp" )
```

Launch-on-Hardware

- launchd runs your job when matching device appears

```
EnableTransactions = 1
Label = "com.mycompany.xyzdevice_attached"
LaunchEvents = {
    "com.apple.iokit.matching" = {
        "xyzdevice-attached" = {
            IOProviderClass = IOUSBDevice
            idProduct = 34050
            idVendor = 1452
        }
    }
}
ProgramArguments = ( "/Applications/MyApp.app/Contents/MacOS/MyApp" )
```

Launch-on-Hardware

- launchd runs your job when matching device appears

```
EnableTransactions = 1
Label = "com.mycompany.xyzdevice_attached"
LaunchEvents = {
    "com.apple.iokit.matching" = {
        "xyzdevice-attached" = {
            IOProviderClass = IOUSBDevice
            idProduct = 34050
            idVendor = 1452
        }
    }
}
ProgramArguments = ( "/Applications/MyApp.app/Contents/MacOS/MyApp" )
```

Launch-on-Hardware

- launchd runs your job when matching device appears

```
EnableTransactions = 1
Label = "com.mycompany.xyzdevice_attached"
LaunchEvents = {
    "com.apple.iokit.matching" = {
        "xyzdevice-attached" = {
            IOProviderClass = IOUSBDevice
            idProduct = 34050
            idVendor = 1452
        }
    }
}
ProgramArguments = ( "/Applications/MyApp.app/Contents/MacOS/MyApp" )
```

See `man launchd.plist`

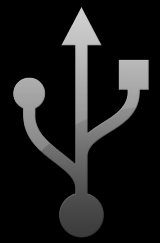
Discovering Hardware

- Discovering IOKit devices in the IORegistry
  - Specify device using matching dictionary
  - Call `IOServiceGetMatchingServices()` to find attached devices
  - Iterate through results
- Registering for notification of new devices
  - Call `IOServiceAddMatchingNotification()` to register callback
- Connecting to devices using an IOUserClient
  - Call `IOCreatePlugInInterfaceForService()` to load device plugin
  - May require admin privilege and/or entitlement if SandBoxed

Tour of Available APIs

- Mac OS X Lion offers a variety of device access APIs
- Bus-level
  - USB, Bluetooth, FireWire, SCSI, Serial
- Service-level
  - Audio, Video, Storage, HID, Scanner & Camera, Printing

USB: User-Space Access

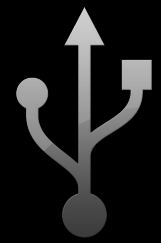


- Use IORegistry and IOUSBLib to access USB devices and interfaces

  Classes `IOUSBDevice` and `IOUSBInterface`
- SandBox access to USB devices

  Entitlement: `com.apple.security.device.usb`
- Apps cannot access an in-use USB device
  - Class drivers such as HID driver can "get in the way"
  - Install a "codeless KEXT" to prevent interference

USB: More Information



USB, Bluetooth, and FireWire Lab	Core OS Lab B Thursday 2:00PM

Technical Q&A QA1370:
Common QA and Roadmap
for USB Software Development
on Mac OS X

USBPrivateDataSample
SampleUSBMIDIDriver

usb@lists.apple.com

`IOKit.framework/usb`

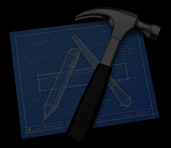
Bluetooth: User-Space Access

- Use specific APIs to interact with Bluetooth devices
  - Use `IOBluetoothDeviceInquiry` to discover devices
  - Use `IOBluetoothSDPServiceRecord` to discover individual services
  - Use `IOBluetoothDevice` and `IOBluetoothL2CAPChannel` for device I/O
  - Note: Objective C APIs preferred
- SandBox access not allowed for Bluetooth devices
- Apps can publish new Bluetooth services
  - Appears as a new service of the Mac
  - Persistent service will launch app if needed to handle remote requests
- Bluetooth services can usually be shared by several apps

Bluetooth: More Information

USB, Bluetooth, and FireWire Lab	Core OS Lab B Thursday 2:00PM

Bluetooth Device Access Guide



bluetooth-dev@lists.apple.com

`IOBluetooth.framework`
`IOBluetoothUI.framework`

FireWire: User-Space Access

- Use IORegistry to interact with FireWire devices
  - Generic classes: `IOFireWireUnit`, `IOFireWireDevice`
  - Storage classes: `IOFireWireSBP2Target`, `IOFireWireSBP2LUN`
  - Should use `IOSCSITaskUserClient` instead
  - A/V class: `IOFireWireAVCUnit`
- SandBox access not allowed for FireWire devices
- Apps cannot access an in-use FireWire device

FireWire: More Information

USB, Bluetooth, and FireWire Lab	Core OS Lab B Thursday 2:00PM

Accessing FireWire Devices From Applications

simpleAVC

firewire@lists.apple.com

`IOKit.framework/firewire`

SCSI: User-Space Access

- Use IORegistry to interact with SCSI Task devices (STUC)
  - Classes with property `SCSITaskDeviceCategory`
- SandBox access not allowed for SCSI devices
- Apps cannot access an in-use SCSI device
  - Exception for authoring devices (e.g., DVD Burner)

SCSI: More Information

USB, Bluetooth, and FireWire Lab	Core OS Lab B Thursday 2:00PM
FileSystems Lab	Core OS Lab A Wednesday 4:30PM

SCSI Architecture Model
Device Interface Guide

STUCAuthoringDeviceTool
STUCOtherDeviceTool

ata-scsi-dev@lists.apple.com

`IOKit.framework/scsi`

Serial: User-Space Access

- Use IORegistry to discover serial devices
- Use POSIX APIs to access serial devices
  - Use `/dev/tty*` device node for incoming connections
  - Use `/dev/cu*` device node for outgoing connections
- SandBox access not allowed for serial interfaces
- Apps cannot access an in-use serial device

Serial: More Information

USB, Bluetooth, and FireWire Lab	Core OS Lab B Thursday 2:00PM
Network Lab	Core OS Lab A Thursday 4:30PM

Device File Access Guide
For Serial Devices

`man termios`

SerialPortSample

darwin-dev@lists.apple.com

`System.framework`

`IOKit.framework/serial`

Audio: User-Space Access

- Use CoreAudio APIs to interact with audio devices
  - Output Audio Unit (AUHAL) wraps HAL's API
  - Use AudioComponent APIs to load AudioUnits
- SandBox access allowed for microphone
  - Entitlement: `com.apple.security.device.microphone`
- Audio interfaces can be published via AudioDriverPlugin
  - Requires admin privileges to install
  - Accessible to other clients of CoreAudio
  - App must stay running while service is published
- All audio interfaces may be shared by multiple apps

Audio: More Information

Music in iOS and Lion	Marina Wednesday 10:15AM
Audio Lab	Graphics, Media & Games Lab C Wednesday 2:00PM

CoreAudio Overview

HALExamples

SampleDriverPlugIn

Apple Developer Forums

http://devforums.apple.com

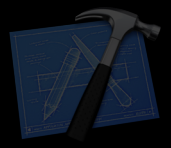
`CoreAudio.framework`

Video: User-Space Access

- Use CoreMedia APIs to interact with video devices
  - Device Abstraction Layer (DAL)
- SandBox access allowed for camera
  - Entitlement: `com.apple.security.device.camera`
- Admin-privileged apps can publish new video interfaces
  - Accessible to other clients of CoreMedia
  - App must stay running while service is published

Video: More Information

AV Foundation Lab	Graphics, Media & Games Lab B Thursday 9:00AM



Apple Developer Forums
http://devforums.apple.com

`CoreMedia.framework`
`CoreMediaIO.framework`

Storage: User-Space Access

- Use DiskArbitration APIs to discover storage devices
  - Use `DARegisterDiskAppearedCallback()` to discover storage devices
  - Alternatively, use IORegistry to discover IOMedia objects
- Apps use BSD APIs to interact with storage devices
  - Use `/dev/rdisk*` for block-level I/O
  - BSD access semantics enforced
- SandBox access not allowed for storage devices

Storage: More Information

FileSystems Lab	Core OS Lab A Wednesday 4:30PM

Disk Arbitration Framework Reference

FSMegaInfo

ata-scsi-dev@lists.apple.com

darwin-dev@lists.apple.com

`System.framework`

`DiskArbitration.framework`

HID: User-Space Access

- Use the IOHIDManager to locate HID devices
  - Use `IOHIDManagerCopyDevices()` to discover HID devices
  - Use `IOHIDDeviceOpen()` to connect to an HID device
- SandBox access not allowed for HID devices
- Apps can "seize" nonkeyboard HID devices
- Apps can post virtual HID events via CoreGraphics

HID: More Information

USB, Bluetooth, and FireWire	Core OS Lab B Thursday 2:00PM

HID Class Device Interface Guide

HID Utilities

usb@lists.apple.com

`IOKit.framework/hid*`

Image Capture: User-Space Access

- Use ImageCaptureCore to interact with cameras and scanners
  - Use `ICDeviceBrowser` to discover image-capture devices
  - Use `requestDownloadFile` to download images
- Use ImageKit to access ImageCapture devices from UI classes
- SandBox access not allowed for ImageCapture devices
- Apps can publish new image-capture devices
  - Appears as a new service of the Mac
  - App must stay running while service is published

Image Capture: More Information

Image Capture Lab	Graphics, Media & Games Lab D Tuesday 11:30AM

Image Capture Applications Programming Guide

VirtualScanner

Apple Developer Forums
http://devforums.apple.com

`ImageCaptureCore.framework`

Printing: User-Space Access

- Use CorePrinting to access printers
- SandBox access allowed for printers
  - Entitlement: `com.apple.security.print`
- Printers published via user-space printer driver
  - Use CUPS APIs to create "backend" driver
  - Appears as new printing service of the Mac
  - Can be shared over network
  - Requires admin privileges to install

Printing: More Information

Printing on iOS and Mac OS X Lab	Application Frameworks Lab D Tuesday 4:30PM

http://www.cups.org

man cups-config

SampleRaster

printing@lists.apple.com

`/usr/include/cups/*`

HID Driver

Turning lead into gold

Thane Norton
I/O HID Team Lead

What You Will Learn

- How to find and access HID devices
- How to inject events

New Hardware!

- Looks cool!
- No driver?

New Hardware!

New Hardware!

New Hardware!

New Hardware!

RGB Backlight

New Hardware!

Where Do I Start?

- HID device
- Use USB Prober

```
Full Speed device @ 13 (0xFD310000): ....  Composite device: "G13"
  Port Information:   0x0018
  Number Of Endpoints (includes EP0):
  Device Descriptor
    Descriptor Version Number:   0x0200
    Device Class:   0   (Composite)
    Device Subclass:   0
    Device Protocol:   0
    Device MaxPacketSize:   8
    Device VendorID/ProductID:   0x046D/0xC21C   (Logitech Inc.)
    Device Version Number:   0x0103
    Number of Configurations:   1
    Manufacturer String:   0 (none)
    Product String:   1 "G13"
    Serial Number String:   0 (none)
  Configuration Descriptor (current config)
    Length (and contents):   41
    Number of Interfaces:   1
    Configuration Value:   1
    Attributes:   0x80 (bus-powered)
    MaxPower:   500 ma
    Interface #0 - HID
      Alternate Setting   0
      Number of Endpoints   2
      Interface Class:   3   (HID)
      Interface Subclass;   0
      Interface Protocol:   0
      HID Descriptor
        Descriptor Version Number:   0x0111
        Country Code:   0
        Descriptor Count:   1
        Descriptor 1
          Type:   0x22  (Report Descriptor)
          Length (and contents):   61
          Parsed Report Descriptor:
              Usage Page  (Vendor defined 0)
              Usage 0 (0x0)
                Collection (Application)
                Logical Minimum.. (0)
                Logical Maximum.. (255)
                Usage 1 (0x1)
                ReportID......... (1)
                Report Count..... (7)
                Report Size...... (8)
                Input............ (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (3)
                Usage 2 (0x2)
                Report Count..... (991)
                Output........... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (7)
                Usage 3 (0x3)
                Report Count..... (4)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (4)
                Usage 4 (0x4)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (5)
                Usage 5 (0x5)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (6)
                Usage 6 (0x6)
                Report Count..... (257)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                End Collection
      Endpoint 0x81 - Interrupt Input
      Endpoint 0x02 - Interrupt Output
```

Where Do I Start?

- HID device
- Use USB Prober

```
Device VendorID/ProductID:    0x046D/0xC21C
```

```
Full Speed device @ 13 (0xFD310000): ....  Composite device: "G13"
  Port Information:    0x0018
  Number Of Endpoints (includes EP0):
  Device Descriptor
    Descriptor Version Number:    0x0200
    Device Class:    0    (Composite)
    Device Subclass:    0
    Device Protocol:    0
    Device MaxPacketSize:    8
    Device VendorID/ProductID:    0x046D/0xC21C    (Logitech Inc.)
    Device Version Number:    0x0103
    Number of Configurations:    1
    Manufacturer String:    0 (none)
    Product String:    1 "G13"
    Serial Number String:    0 (none)
  Configuration Descriptor (current config)
    Length (and contents):    41
    Number of Interfaces:    1
    Configuration Value:    1
    Attributes:    0x80 (bus-powered)
    MaxPower:    500 ma
    Interface #0 - HID
      Alternate Setting    0
      Number of Endpoints    2
      Interface Class:    3    (HID)
      Interface Subclass;    0
      Interface Protocol:    0
      HID Descriptor
        Descriptor Version Number:    0x0111
        Country Code:    0
        Descriptor Count:    1
        Descriptor 1
          Type:    0x22  (Report Descriptor)
          Length (and contents):    61
          Parsed Report Descriptor:
              Usage Page  (Vendor defined 0)
              Usage 0 (0x0)
                Collection (Application)
                Logical Minimum.. (0)
                Logical Maximum.. (255)
                Usage 1 (0x1)
                ReportID......... (1)
                Report Count..... (7)
                Report Size...... (8)
                Input............ (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (3)
                Usage 2 (0x2)
                Report Count..... (991)
                Output........... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (7)
                Usage 3 (0x3)
                Report Count..... (4)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (4)
                Usage 4 (0x4)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (5)
                Usage 5 (0x5)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (6)
                Usage 6 (0x6)
                Report Count..... (257)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                End Collection
      Endpoint 0x81 - Interrupt Input
      Endpoint 0x02 - Interrupt Output
```

Where Do I Start?

- HID device
- Use USB Prober

```
Parsed Report Descriptor:
  Usage Page  (Vendor defined 0)
  Usage 0 (0x0)
    Collection (Application)
    Logical Minimum.. (0)
    Logical Maximum.. (255)
    Usage 1 (0x1)
    ReportID......... (1)
    Report Count..... (7)
    Report Size...... (8)
```

```
Full Speed device @ 13 (0xFD310000): ....  Composite device: "G13"
  Port Information:   0x0018
  Number Of Endpoints (includes EP0):
  Device Descriptor
    Descriptor Version Number:   0x0200
    Device Class:   0   (Composite)
    Device Subclass:   0
    Device Protocol:   0
    Device MaxPacketSize:   8
    Device VendorID/ProductID:   0x046D/0xC21C   (Logitech Inc.)
    Device Version Number:   0x0103
    Number of Configurations:   1
    Manufacturer String:   0 (none)
    Product String:   1 "G13"
    Serial Number String:   0 (none)
  Configuration Descriptor (current config)
    Length (and contents):   41
    Number of Interfaces:   1
    Configuration Value:   1
    Attributes:   0x80 (bus-powered)
    MaxPower:   500 ma
    Interface #0 - HID
      Alternate Setting   0
      Number of Endpoints   2
      Interface Class:   3   (HID)
      Interface Subclass;   0
      Interface Protocol:   0
      HID Descriptor
        Descriptor Version Number:   0x0111
        Country Code:   0
        Descriptor Count:   1
        Descriptor 1
          Type:   0x22  (Report Descriptor)
          Length (and contents):   61
          Parsed Report Descriptor:
              Usage Page  (Vendor defined 0)
              Usage 0 (0x0)
                Collection (Application)
                Logical Minimum.. (0)
                Logical Maximum.. (255)
                Usage 1 (0x1)
                ReportID......... (1)
                Report Count..... (7)
                Report Size...... (8)
                Input............ (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (3)
                Usage 2 (0x2)
                Report Count..... (991)
                Output........... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (7)
                Usage 3 (0x3)
                Report Count..... (4)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (4)
                Usage 4 (0x4)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (5)
                Usage 5 (0x5)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                ReportID......... (6)
                Usage 6 (0x6)
                Report Count..... (257)
                Feature.......... (Data, Variable, Absolute, No Wrap, Linear,
                End Collection
      Endpoint 0x81 - Interrupt Input
      Endpoint 0x02 - Interrupt Output
```

Where Do I Start?

- Developer documentation
- Open source drivers
- “Play with it”
- Packet inspection
  - Strictly for the hard core

Writing a Driver

A sample project

Error Handling

- Use a consistent idiom
- May want to use AssertMacros.h
- Definitely want to use asl.h

  See `man asl` for more info

Device Discovery

```
_hidManager = IOHIDManagerCreate(NULL, 0);
CFDictionaryRef matching = (CFDictionaryRef)
    [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:0x046d], @kIOHIDVendorIDKey,
            [NSNumber numberWithInt:0xC21C], @kIOHIDProductKey, Nil];
IOHIDManagerSetDeviceMatching(_hidManager, matching);
IOHIDManagerRegisterDeviceMatchingCallback(_hidManager,
        SimpleG13DriverAppDelegateDeviceCallback, self);
IOHIDManagerScheduleWithRunLoop(_hidManager,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
IOReturn err = IOHIDManagerOpen(_hidManager, 0);
check_noerr(err);
```

Device Discovery

```
_hidManager = IOHIDManagerCreate(NULL, 0);
CFDictionaryRef matching = (CFDictionaryRef)
    [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:0x046d], @kIOHIDVendorIDKey,
            [NSNumber numberWithInt:0xC21C], @kIOHIDProductKey, Nil];
IOHIDManagerSetDeviceMatching(_hidManager, matching);
IOHIDManagerRegisterDeviceMatchingCallback(_hidManager,
        SimpleG13DriverAppDelegateDeviceCallback, self);
IOHIDManagerScheduleWithRunLoop(_hidManager,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
IOReturn err = IOHIDManagerOpen(_hidManager, 0);
check_noerr(err);
```

Device Discovery

```
_hidManager = IOHIDManagerCreate(NULL, 0);
CFDictionaryRef matching = (CFDictionaryRef)
    [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:0x046d], @kIOHIDVendorIDKey,
            [NSNumber numberWithInt:0xC21C], @kIOHIDProductKey, Nil];
IOHIDManagerSetDeviceMatching(_hidManager, matching);
IOHIDManagerRegisterDeviceMatchingCallback(_hidManager,
        SimpleG13DriverAppDelegateDeviceCallback, self);
IOHIDManagerScheduleWithRunLoop(_hidManager,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
IOReturn err = IOHIDManagerOpen(_hidManager, 0);
check_noerr(err);
```

Device Discovery

```
_hidManager = IOHIDManagerCreate(NULL, 0);
CFDictionaryRef matching = (CFDictionaryRef)
    [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:0x046d], @kIOHIDVendorIDKey,
            [NSNumber numberWithInt:0xC21C], @kIOHIDProductKey, Nil];
IOHIDManagerSetDeviceMatching(_hidManager, matching);
IOHIDManagerRegisterDeviceMatchingCallback(_hidManager,
        SimpleG13DriverAppDelegateDeviceCallback, self);
IOHIDManagerScheduleWithRunLoop(_hidManager,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
IOReturn err = IOHIDManagerOpen(_hidManager, 0);
check_noerr(err);
```

Device Discovery

```
_hidManager = IOHIDManagerCreate(NULL, 0);
CFDictionaryRef matching = (CFDictionaryRef)
    [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:0x046d], @kIOHIDVendorIDKey,
            [NSNumber numberWithInt:0xC21C], @kIOHIDProductKey, Nil];
IOHIDManagerSetDeviceMatching(_hidManager, matching);
IOHIDManagerRegisterDeviceMatchingCallback(_hidManager,
        SimpleG13DriverAppDelegateDeviceCallback, self);
IOHIDManagerScheduleWithRunLoop(_hidManager,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
IOReturn err = IOHIDManagerOpen(_hidManager, 0);
check_noerr(err);
```

Device Discovery

```
_hidManager = IOHIDManagerCreate(NULL, 0);
CFDictionaryRef matching = (CFDictionaryRef)
    [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:0x046d], @kIOHIDVendorIDKey,
            [NSNumber numberWithInt:0xC21C], @kIOHIDProductKey, Nil];
IOHIDManagerSetDeviceMatching(_hidManager, matching);
IOHIDManagerRegisterDeviceMatchingCallback(_hidManager,
        SimpleG13DriverAppDelegateDeviceCallback, self);
IOHIDManagerScheduleWithRunLoop(_hidManager,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
IOReturn err = IOHIDManagerOpen(_hidManager, 0);
check_noerr(err);
```

Device Discovery

```
_hidManager = IOHIDManagerCreate(NULL, 0);
CFDictionaryRef matching = (CFDictionaryRef)
    [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:0x046d], @kIOHIDVendorIDKey,
            [NSNumber numberWithInt:0xC21C], @kIOHIDProductKey, Nil];
IOHIDManagerSetDeviceMatching(_hidManager, matching);
IOHIDManagerRegisterDeviceMatchingCallback(_hidManager,
        SimpleG13DriverAppDelegateDeviceCallback, self);
IOHIDManagerScheduleWithRunLoop(_hidManager,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
IOReturn err = IOHIDManagerOpen(_hidManager, 0);
check_noerr(err);
```

Device Discovery

```
_hidManager = IOHIDManagerCreate(NULL, 0);
CFDictionaryRef matching = (CFDictionaryRef)
    [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:0x046d], @kIOHIDVendorIDKey,
            [NSNumber numberWithInt:0xC21C], @kIOHIDProductKey, Nil];
IOHIDManagerSetDeviceMatching(_hidManager, matching);
IOHIDManagerRegisterDeviceMatchingCallback(_hidManager,
        SimpleG13DriverAppDelegateDeviceCallback, self);
IOHIDManagerScheduleWithRunLoop(_hidManager,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
IOReturn err = IOHIDManagerOpen(_hidManager, 0);
check_noerr(err);
```

Device Discovery

```
_hidManager = IOHIDManagerCreate(NULL, 0);
CFDictionaryRef matching = (CFDictionaryRef)
    [NSDictionary dictionaryWithObjectsAndKeys:
            [NSNumber numberWithInt:0x046d], @kIOHIDVendorIDKey,
            [NSNumber numberWithInt:0xC21C], @kIOHIDProductKey, Nil];
IOHIDManagerSetDeviceMatching(_hidManager, matching);
IOHIDManagerRegisterDeviceMatchingCallback(_hidManager,
        SimpleG13DriverAppDelegateDeviceCallback, self);
IOHIDManagerScheduleWithRunLoop(_hidManager,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes);
IOReturn err = IOHIDManagerOpen(_hidManager, 0);
check_noerr(err); // <-- PLEASE CHECK ERRORS
```

Device Found

```
IOHIDDeviceRegisterRemovalCallback(newDevice,
        SimpleG13DriverAppDelegateRemovalCallback, self);
IOReturn err = IOHIDDeviceOpen(_device, kIOHIDOptionsTypeSeizeDevice);
check_noerr(err);
IOHIDDeviceRegisterInputValueCallback(_device,
        SimpleG13DriverDeviceValueCallback, self);
IOHIDDeviceScheduleWithRunLoop(_device, CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);
```

Device Found

```
IOHIDDeviceRegisterRemovalCallback(newDevice,
        SimpleG13DriverAppDelegateRemovalCallback, self);
IOReturn err = IOHIDDeviceOpen(_device, kIOHIDOptionsTypeSeizeDevice);
check_noerr(err);
IOHIDDeviceRegisterInputValueCallback(_device,
        SimpleG13DriverDeviceValueCallback, self);
IOHIDDeviceScheduleWithRunLoop(_device, CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);
```

Device Found

```
IOHIDDeviceRegisterRemovalCallback(newDevice,
        SimpleG13DriverAppDelegateRemovalCallback, self);
IOReturn err = IOHIDDeviceOpen(_device, kIOHIDOptionsTypeSeizeDevice);
check_noerr(err);
IOHIDDeviceRegisterInputValueCallback(_device,
        SimpleG13DriverDeviceValueCallback, self);
IOHIDDeviceScheduleWithRunLoop(_device, CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);
```

Device Found

```
IOHIDDeviceRegisterRemovalCallback(newDevice,
        SimpleG13DriverAppDelegateRemovalCallback, self);
IOReturn err = IOHIDDeviceOpen(_device, 0);
check_noerr(err);
IOHIDDeviceRegisterInputValueCallback(_device,
        SimpleG13DriverDeviceValueCallback, self);
IOHIDDeviceScheduleWithRunLoop(_device, CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);
```

Device Found

```
IOHIDDeviceRegisterRemovalCallback(newDevice,
        SimpleG13DriverAppDelegateRemovalCallback, self);
IOReturn err = IOHIDDeviceOpen(_device, 0);
check_noerr(err); // <-- PLEASE CHECK ERRORS
IOHIDDeviceRegisterInputValueCallback(_device,
        SimpleG13DriverDeviceValueCallback, self);
IOHIDDeviceScheduleWithRunLoop(_device, CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);
```

Device Found

```
IOHIDDeviceRegisterRemovalCallback(newDevice,
        SimpleG13DriverAppDelegateRemovalCallback, self);
IOReturn err = IOHIDDeviceOpen(_device, 0);
check_noerr(err);
IOHIDDeviceRegisterInputValueCallback(_device,
        SimpleG13DriverDeviceValueCallback, self);
IOHIDDeviceScheduleWithRunLoop(_device, CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);
```

Device Found

```
IOHIDDeviceRegisterRemovalCallback(newDevice,
        SimpleG13DriverAppDelegateRemovalCallback, self);
IOReturn err = IOHIDDeviceOpen(_device, 0);
check_noerr(err);
IOHIDDeviceRegisterInputValueCallback(_device,
        SimpleG13DriverDeviceValueCallback, self);
IOHIDDeviceScheduleWithRunLoop(_device, CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);
```

Device Found

```
IOHIDDeviceRegisterRemovalCallback(newDevice,
        SimpleG13DriverAppDelegateRemovalCallback, self);
IOReturn err = IOHIDDeviceOpen(_device, 0);
check_noerr(err);
IOHIDDeviceRegisterInputValueCallback(_device,
        SimpleG13DriverDeviceValueCallback, self);
IOHIDDeviceScheduleWithRunLoop(_device, CFRunLoopGetCurrent(),
        kCFRunLoopCommonModes);
```

- Device reference is retained by the manager

Device Data

- I got data from the device!
- Generate events

```
CGEventSourceRef _source = CGEventSourceCreate(
    kCGEventSourceStateCombinedSessionState
);
CGEventSourceSetKeyboardType(_source, 46);
```

Device Data

- I got data from the device!
- Generate events

```
CGEventSourceRef _source = CGEventSourceCreate(
    kCGEventSourceStateCombinedSessionState
);
CGEventSourceSetKeyboardType(_source, 46);
```

Device Data

- I got data from the device!
- Generate events

```
CGEventSourceRef _source = CGEventSourceCreate(
    kCGEventSourceStateCombinedSessionState
);
CGEventSourceSetKeyboardType(_source, 46);
```

- HIDSubinterfaceID
- CGEventSourceGetKeyboardType

Device Data—Joystick


```
0xC0
0xDE
0xF0
0x0D
0xCA
0xFE
0xBE
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventMouseMoved, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventMouseMoved, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventMouseMoved, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventMouseMoved, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventMouseMoved, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventMouseMoved, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventMouseMoved, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventMouseMoved, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventLeftMouseDown, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventLeftMouseDragged, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Joystick

Mouse movement

```
CGPoint position;
position.x = current.x + (data[0] - 128.0) / 10.0;
position.y = current.y + (data[1] - 128.0) / 10.0;
CGEventRef event = CGEventCreateMouseEvent(
        _source, kCGEventLeftMouseUp, position, 0);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys


```
0xC0
0xDE
0xF0
0x0D
0xCA
0xFE
0xBE
```

Device Data—G Keys

Keyboard events

```
CGEventRef event = CGEventCreateKeyboardEvent(
        _source, virtualKeyCode, true);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Keyboard events

```
CGEventRef event = CGEventCreateKeyboardEvent(
        _source, virtualKeyCode, true);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Keyboard events

```
CGEventRef event = CGEventCreateKeyboardEvent(
        _source, virtualKeyCode, true);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Keyboard events

```
CGEventRef event = CGEventCreateKeyboardEvent(
        _source, virtualKeyCode, false);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Keyboard events

```
CGEventRef event = CGEventCreateKeyboardEvent(
        _source, virtualKeyCode, false);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Keyboard events

```
CGEventRef event = CGEventCreateKeyboardEvent(
        _source, virtualKeyCode, false);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

- See <HIToolbox/Events.h> for first-order approximations
- Gather data from real keyboards

Device Data—G Keys

Keyboard events

```
CGEventRef event = CGEventCreateKeyboardEvent(
        _source, virtualKeyCode, false);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Keyboard events

```
CGEventRef event = CGEventCreateKeyboardEvent(
        _source, virtualKeyCode, false);
require(event, complain);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Modifier keys

```
CGEventRef event = CGEventCreate(_source);
require(event, complain);
CGEventSetType(event, kCGEventFlagsChanged);
modifierState |= kCGEventFlagMaskAlphaShift;
CGEventSetFlags(event, modifierState);
CGEventSetIntegerValueField(event,
        kCGKeyboardEventKeycode, virtualKeyCode);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Modifier keys

```
CGEventRef event = CGEventCreate(_source);
require(event, complain);
CGEventSetType(event, kCGEventFlagsChanged);
modifierState |= kCGEventFlagMaskAlphaShift;
CGEventSetFlags(event, modifierState);
CGEventSetIntegerValueField(event,
        kCGKeyboardEventKeycode, virtualKeyCode);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Modifier keys

```
CGEventRef event = CGEventCreate(_source);
require(event, complain);
CGEventSetType(event, kCGEventFlagsChanged);
modifierState |= kCGEventFlagMaskAlphaShift;
CGEventSetFlags(event, modifierState);
CGEventSetIntegerValueField(event,
        kCGKeyboardEventKeycode, virtualKeyCode);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Modifier keys

```
CGEventRef event = CGEventCreate(_source);
require(event, complain);
CGEventSetType(event, kCGEventFlagsChanged);
modifierState |= kCGEventFlagMaskAlphaShift;
CGEventSetFlags(event, modifierState);
CGEventSetIntegerValueField(event,
        kCGKeyboardEventKeycode, virtualKeyCode);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Modifier keys

```
CGEventRef event = CGEventCreate(_source);
require(event, complain);
CGEventSetType(event, kCGEventFlagsChanged);
modifierState |= kCGEventFlagMaskAlphaShift;
CGEventSetFlags(event, modifierState);
CGEventSetIntegerValueField(event,
        kCGKeyboardEventKeycode, virtualKeyCode);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—G Keys

Modifier keys

```
CGEventRef event = CGEventCreate(_source);
require(event, complain);
CGEventSetType(event, kCGEventFlagsChanged);
modifierState |= kCGEventFlagMaskAlphaShift;
CGEventSetFlags(event, modifierState);
CGEventSetIntegerValueField(event,
        kCGKeyboardEventKeycode, virtualKeyCode);
CGEventPost(kCGSessionEventTap, event);
CFRelease(event);
complain:
    // complain
```

Device Data—Other Keys


```
0xC0
0xDE
0xF0
0x0D
0xCA
0xFE
0xBE
```

Device Data—Other Keys

- Internal-state change
  - Reflect that change back to the hardware

Device Data—Other Keys

Backlight color

```
uint8_t data[4];
data[0] = 0;
data[1] = [color redComponent] * 255;
data[2] = [color greenComponent] * 255;
data[3] = [color blueComponent] * 255;
IOReturn err = IOHIDDeviceSetReport(_device,
        kIOHIDReportTypeFeature, 7,
        data, sizeof(data));
check_noerr(err);
```

Device Data—Other Keys

Backlight color

```
uint8_t data[4];
data[0] = 0;
data[1] = [color redComponent] * 255;
data[2] = [color greenComponent] * 255;
data[3] = [color blueComponent] * 255;
IOReturn err = IOHIDDeviceSetReport(_device,
        kIOHIDReportTypeFeature, 7,
        data, sizeof(data));
check_noerr(err);
```

Device Data—Other Keys

Backlight color

```
uint8_t data[4];
data[0] = 0;
data[1] = [color redComponent] * 255;
data[2] = [color greenComponent] * 255;
data[3] = [color blueComponent] * 255;
IOReturn err = IOHIDDeviceSetReport(_device,
        kIOHIDReportTypeFeature, 7,
        data, sizeof(data));
check_noerr(err);
```

Device Data—Other Keys

LEDs

```
uint8_t data[4] = {0};
data[1] = lights & 0xf;
IOReturn err = IOHIDDeviceSetReport(_device,
        kIOHIDReportTypeFeature, 5,
        data, sizeof(data));
check_noerr(err);
```

Device Data—Other Keys

LEDs

```
uint8_t data[4] = {0};
data[1] = lights & 0xf;
IOReturn err = IOHIDDeviceSetReport(_device,
        kIOHIDReportTypeFeature, 5,
        data, sizeof(data));
check_noerr(err);
```

Device Data—Other Keys

LEDs

```
uint8_t data[4] = {0};
data[1] = lights & 0xf;
IOReturn err = IOHIDDeviceSetReport(_device,
        kIOHIDReportTypeFeature, 5,
        data, sizeof(data));
check_noerr(err);
```

Device Detach

- Your callback is called
  - Call `IOHIDDeviceClose()`
- Do not talk to device
- Do not need to unregister

Driver Quit

- Call `IOHIDManagerClose()`
- Release the manager reference
- Fin

Summary

- Use an IOHIDManager to track devices
- Use an IOHIDDevice to
  - Get data from a device
  - Send data to a device
    - One physical device may publish multiple HID devices
- Check your errors
- Writing a driver for a HID device is easy

Userland Device Access Lab	Core OS Lab B Thursday 9:00AM

Sleep/Wake for Apps

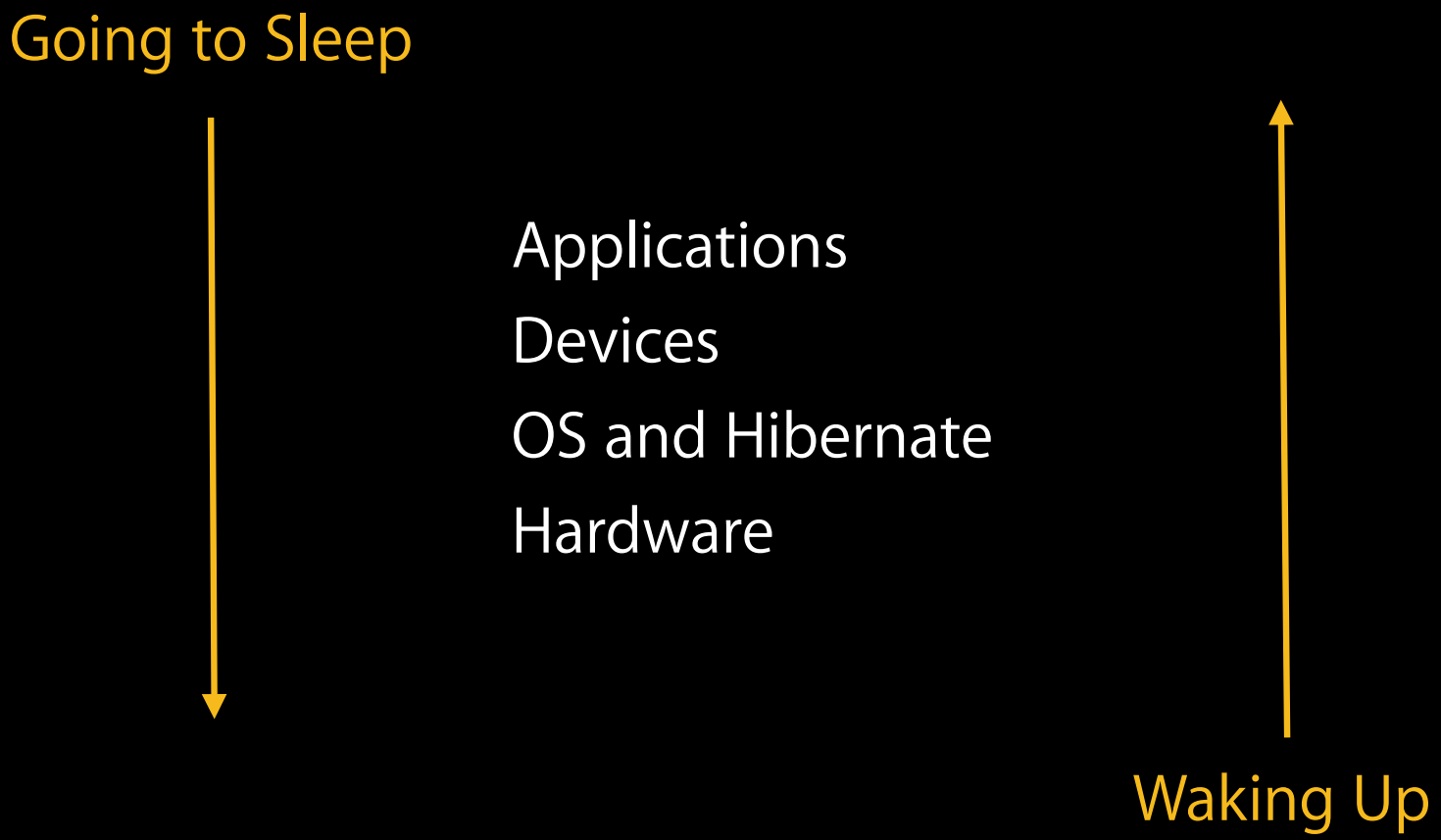
Ethan Bold
I/O Kit Team

Sleep and Wake

Only on Mac OS

- Best practices for sleep/wake API
- New behaviors in sleep/wake

Sleep and Wake



Listen for Sleep/Wake Notifications

`IORegisterForSystemPower()`

- Lets you run code before sleep and after wake
- Sleep handler must acknowledge ASAP
  - Call `IOAllowPowerChange()`

Register and Deregister

CFRunLoop

- At launch

```
IORegisterForSystemPower();
IONotificationPortCreateRunLoopSource();
CFRunLoopAddSource();
```

- At exit

```
IODeregisterForSystemPower();
```

Register and Deregister

Dispatch source

- At launch

```
IORegisterForSystemPower();
IONotificationPortCreateDispatchSource();
dispatch_queue_add_source();
```

- At exit

```
IODeregisterForSystemPower();
```

Implementing a Sleep/Wake Handler

```
MySleepWakeHandler(..., uint32_t msgType, void *handlerID)
{
    if (msgType == kIOMessageSystemHasPoweredOn) {
        // Do wake-up work
    }
    if (msgType == kIOMessageSystemWillSleep) {
        // Do Going To Sleep Work
        IOAllowPowerChange(PMRoot, handlerID);
    }
    if (msgType == kIOMessageCanSystemSleep) {
        IOAllowPowerChange(PMRoot, handlerID);
    }
}
```

Power Assertions

- API to keep the system and the display awake

  `IOPMAssertionCreateWithName()`

- Takes arguments

  `kIOPMAssertionTypePreventUserIdleDisplaySleep`

  `kIOPMAssertionTypePreventUserIdleSleep`

- Use power assertions when you are doing important work

- More accountable

Power Assertion Sample Code

```
IOPMAssertionID newAssertionID;

IOPMAssertionCreateWithName(
     kIOPMAssertionTypePreventUserIdleSleep,
     kIOPMAssertionLevelOn,
     CFSTR(“com.mycompany.myapp.description”),
     &newAssertionID);

...
IOPMAssertionRelease(newAssertionID);
```

Great Command-Line Tools

- caffeinate
- pmset -g assertions
- pmset -g log

New Types of Sleep/Wake

- Standby—new behavior in 10.6.7 to extend battery life while sleeping
- DarkWake—new behavior in Lion to perform work while the computer appears to sleep

Standby

- After sleeping for about an hour, OS X writes memory to a hibernation image on mass storage
- 10.6.7 and Lion
- Only latest MacBook Air models

DarkWake

- The CPU and device drivers power on, but the display and audio remain off

- OS X might enter DarkWake when
  - Attaching external devices
  - Maintaining a network connection (DHCP, Back to My Mac)
  - File sharing, printer sharing, iTunes sharing

Consequences to the Developer

- DarkWake and Standby are not exposed in OS X API
- Clients of `IORegisterForSystemPower()` will not receive notifications during DarkWake or Standby

Take-Aways

- Sleep/wake notifications

  `IORegisterForSystemPower()`

- Asserting a power need

  `IOPMAssertionCreateWithName()`

- Technical Q&A QA1340

Labs

Userland Device Access Lab	Core OS Lab B Thursday 9:00–11:15AM
Power Management Lab	Core OS Lab A Thursday 9:00–11:15AM


WWDC2011