



# [Guide] Creating a Custom SSDT for USBInjectAll.kext

[1](#)[2](#)[3](#)[4](#)[5](#)[6](#)[→](#)[220](#)[Next >](#)

## RehabMan

Moderator

Joined: May 3, 2012  
Messages: 183,006  
Motherboard: Intel DH67BL  
CPU: Core i7-2600K  
Graphics: Intel HD 3000  
Mac:   
Mobile Phone: 

Jan 2, 2017 at 10:12 PM

#1

### Overview

One of the serious issues most users will face with 10.11.x (and later, including 10.12.x) is the new USB stack in OS X/macOS. It has a much heavier reliance on ACPI, and as a result is much more likely to expose bugs in your ACPI implementation of `_PLD`, and `_UPC`.

It is covered in detail in my guide: <https://www.tonymacx86.com/threads/guide-10-11-usb-changes-and-solutions.173616/>

You should be familiar with the writeup above before attempting a custom SSDT.

Many tend to over-simplify the problem, and, for example, believe they can simply apply the port-limit patch and install USBInjectAll.kext and be done with it. In fact, the port limit patch is known to cause problems (it causes access outside a fixed array bounds). Somewhat alarming is the fact that Multibeast includes the port limit patch without mention of its problems.

The fact: Using the port limit patch is not a long term solution. For reliable USB (assuming your ACPI implementation of `_PLD` and `_UPC` is broken... and you don't want to fix it), you must implement a custom SSDT for USBInjectAll.kext that configures your ports on XHC such that the port limit patch is not needed, and each `UsbConnector` value is correct for each port.

Not only does a custom SSDT allow you to avoid the port limit patch, disabling USB ports that are not used, can have some power saving properties, and can avoid bugs with sleep, restart, or shutdown.

This guide will show how this is done.

The process consists of the following steps:

- preparation for port discovery
- port discovery
- creating custom SSDT for USBInjectAll.kext

- testing/verification

The MaciASL used by this guide is available

here: <https://bitbucket.org/RehabMan/os-x-maciasl-patchmatic/downloads/>

Make sure you have ACPI 6.1 selected in MaciASL->Preferences->iASL.

## Preparation for port discovery

In order to create a custom SSDT that contains the correct data for a given computer's USB arrangement, we must first discover all the ports that need to be enabled. Once we know which ports are used, we can eliminate the unused ports.

In order to discover the ports, we need to be certain all ports are enabled.

Requirements:

- EHC1->EH01 and EHC2->EH02 rename (in config.plist), if applicable (your chipset may not have EHCI, or it may be disabled)
- XHCI controller must be named XHC (for most PCs it is default)
- port limit patch (in config.plist)
- install USBInjectAll.kext (install to the system volume)
- if you plan to use it, install FakePCIID.kext + FakePCIID\_XHCIMux.kext. FakePCIID\_XHCIMux only applicable if you have enabled EHCI controller(s).
- if you have an existing SSDT for USBInjectAll, use -uia\_ignore\_rmc
- XHCI injector kext, if required (200-series need XHCI-200-series-injector.kext, 300-series need XHCI-300-series-injector.kext)

The EHCx renames and port limit patches are available in config\_patches.plist ([https://github.com/RehabMan/OS-X-USB-Inject-All/raw/master/config\\_patches.plist](https://github.com/RehabMan/OS-X-USB-Inject-All/raw/master/config_patches.plist)) in the USBInjectAll.kext repository. Use copy/paste from a plist editor to get them into your own config.plist.

The EHCx renames look like this in Xcode:

Key	Type	Value
▼ Root	Dictionary	(2 items)
▼ ACPI	Dictionary	(1 item)
▼ DSDT	Dictionary	(1 item)
▼ Patches	Array	(3 items)
▶ Item 0	Dictionary	(3 items)
▼ Item 1	Dictionary	(3 items)
Comment	String	change EHC1 to EH01
Find	Data	<45484331>
Replace	Data	<45483031>
▼ Item 2	Dictionary	(3 items)
Comment	String	change EHC2 to EH02
Find	Data	<45484332>
Replace	Data	<45483032>

The port limit patches look like this in Xcode:

Key	Type	Value
▼ Root	Dictionary	(2 items)
▶ ACPI	Dictionary	(1 item)
▼ KernelAndKextPatches	Dictionary	(1 item)
▼ KextsToPatch	Array	(4 items)
▶ Item 0	Dictionary	(5 items)
▼ Item 1	Dictionary	(5 items)
Comment	String	change 15 port limit to 26 in XHCI kext (100-series)
MatchOS	String	10.11.x
Name	String	com.apple.driver.usb.AppleUSBXHCIPCI
Find	Data	<83bd8cfe ffff10>
Replace	Data	<83bd8cfe ffff1b>
▶ Item 2	Dictionary	(5 items)
▼ Item 3	Dictionary	(5 items)
Comment	String	change 15 port limit to 26 in XHCI kext (100-series)
MatchOS	String	10.12.x
Name	String	com.apple.driver.usb.AppleUSBXHCIPCI
Find	Data	<83bd74ff ffff10>
Replace	Data	<83bd74ff ffff1b>

The `-uia_ignore_rmc` is added to `config.plist/Boot/Arguments`:

Key	Type	Value
▼ Root	Dictionary	(11 items)
▶ ACPI	Dictionary	(2 items)
▼ Boot	Dictionary	(7 items)
Arguments	String	kext-dev-mode=1 nv_disable=1 dart=0 -uia_ignore_rmc

After preparing for port discovery, you should check in `ioreg` that all ports are injected for your chipset on both EHCI and XHCI controllers.

This is what full injection looks like on my NUC6i7KYK (Skylake). You can see the same ports are injected as provided in the 'ports' dictionary from `USBInjectAll.kext`:

XHC@14

XHC@14000000

HS01@14100000

HS02@14200000

HS03@14300000

IOUSBHostDevice@14300000

AppleUSB20Hub@14300000

AppleUSB20HubPort@14310000

AppleUSB20HubPort@14320000

AppleUSB20HubPort@14330000

AppleUSB20HubPort@14340000

AppleUSBHostLegacyClient

IOUSBHostInterface@0

HS04@14400000

HS05@14500000

HS06@14600000

HS07@14700000

HS08@14800000

HS09@14900000

IOUSBHostDevice@14900000

AppleUSBHostLegacyClient

IOBluetoothHostControllerUSBTransport

IOUSBHostInterface@0

IOUSBHostInterface@1

HS10@14a00000

HS11@14b00000

HS12@14c00000

HS13@14d00000

HS14@14e00000

SS01@15100000

SS02@15200000

SS03@15300000

SS04@15400000

SS05@15500000

SS06@15600000

SS07@15700000

SS08@15800000

SS09@15900000

SS10@15a00000

USR1@14f00000

USR2@15000000

IOGeneralInterestStringIOCommand is n

IOMatchCategoryStringIODefaultMatchC

IOPCIPauseCompatibleBooleanTrue

IOPCIPrimaryMatchString0xa12f8086

IOPCITunnelCompatibleBooleanTrue

IOPowerManagementDictionary6 values

IOProbeScoreNumber0x3e8

IOProviderClassStringIOPCIDevice

kUSBSleepSupportedBooleanTrue

locationIDNumber0x14000000

nameData<"XHC">

port-countData<1a 00 00 00>

portsDictionary26 values

portsHS01Dictionary2 values

portsHS02Dictionary2 values

portsHS03Dictionary2 values

portsHS04Dictionary2 values

portsHS05Dictionary2 values

portsHS06Dictionary2 values

portsHS07Dictionary2 values

portsHS08Dictionary2 values

portsHS09Dictionary2 values

portsHS10Dictionary2 values

portsHS11Dictionary2 values

portsHS12Dictionary2 values

portsHS13Dictionary2 values

portsHS14Dictionary2 values

portsSS01Dictionary2 values

portsSS02Dictionary2 values

portsSS03Dictionary2 values

portsSS04Dictionary2 values

portsSS05Dictionary2 values

portsSS06Dictionary2 values

portsSS07Dictionary2 values

portsSS08Dictionary2 values

portsSS09Dictionary2 values

portsSS10Dictionary2 values

portsUSR1Dictionary2 values

portsUSR2Dictionary2 values

RevisionData<00 03>

RM,USBInjectAllBooleanTrue

And here is what it looks like on my Lenovo u430 using FakePCIID\_XHCIMux.kext.

Note they are the same as the ports provided by USBInjectAll.kext.

XHC:

XHC@14

FakePCIID\_XHCIMux

XHC@14000000

HS01@14100000

HS02@14200000

HS03@14300000

HS04@14400000

HS05@14500000

HS06@14600000

HS07@14700000

HS08@14800000

HS09@14900000

SSP1@14a00000

SSP2@14b00000

SSP3@14c00000

SSP4@14d00000

PDR

PNLF@0

IntelBacklightPanel

PS2B

ps2controller

ApplePS2Controller

ApplePS2KeyboardDevice

ApplePS2Keyboard

IOHIDKeyboardDevice

IOHIDInterface

IOHIDLibUserClient

IOHIDLibUserClient

IOHIDLibUserClient

IOHIDLibUserClient

IOHIDSystem

IOHIDParamUserClient

IOHIDLibUserClient

64bit

Boolean

True

CFBundleIdentifier

String

com.apple.d

controller-statistics

Dictionary

3 values

device-properties

Dictionary

9 values

IOClass

String

AppleUSBX

IOMatchCategory

String

IODefaultMa

IOPCIPrimaryMatch

String

0x9c318086

IOPCITunnelCompatible

Boolean

True

IOPowerManagement

Dictionary

6 values

IOProbeScore

Number

0x3e8

IOProviderClass

String

IOPCIDevice

kUSBSleepPortCurrentLimit

Number

0x834

kUSBSleepSupported

Boolean

True

kUSBWakePortCurrentLimit

Number

0x834

locationID

Number

0x14000000

name

Data

<"XHC">

port-count

Data

<0d 00 00 00>

ports

Dictionary

13 values

HS01

Dictionary

2 values

HS02

Dictionary

2 values

HS03

Dictionary

2 values

HS04

Dictionary

2 values

HS05

Dictionary

2 values

HS06

Dictionary

2 values

HS07

Dictionary

2 values

HS08

Dictionary

2 values

HS09

Dictionary

2 values

SSP1

Dictionary

2 values

SSP2

Dictionary

2 values

SSP3

Dictionary

2 values

SSP4

Dictionary

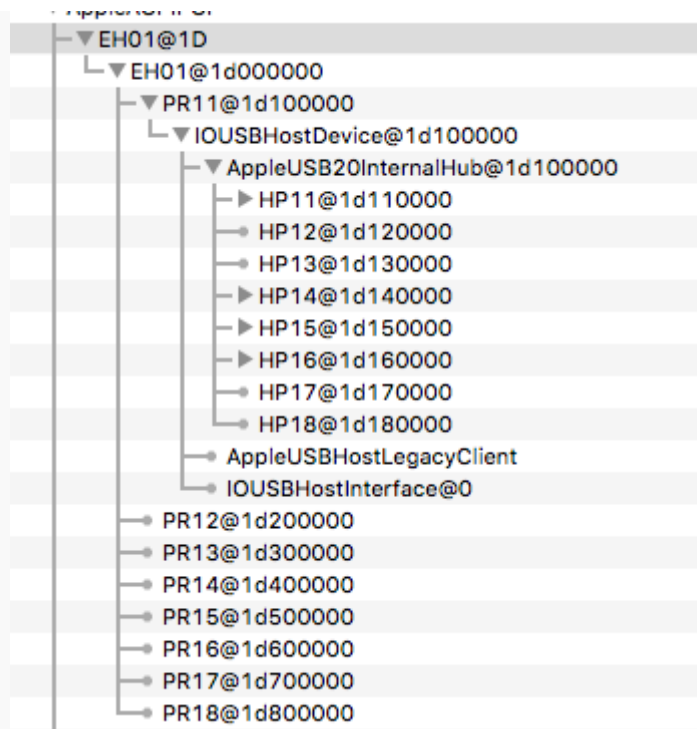
2 values

RM,USBInjectAll

Boolean

True

EH01:



Once you have all ports being injected, you can proceed to the next step where you will determine which ports are actually needed.

### Port discovery

In order to discover which ports need to be in the SSDT, we will use IORegistryExplorer:

- run IORegistryExplorer
- test each port with both USB2 and USB3 devices

Since IORegistryExplorer tracks changes to IOKit objects (existing objects black, new objects in green, disconnected objects in red), we can use the data to determine which ports are actually used.

IORegistryExplorer.app can be downloaded from the attachment at this thread: <http://www.tonymacx86.com/audio/58368-guide-how-make-copy-ioreg.html>

After you download and extract IORegistryExplorer.app, run it. And keep it running throughout the test.

Note: All data needed by this guide is in the IOService registry plane. DO NOT CHANGE the registry plane from IOService.

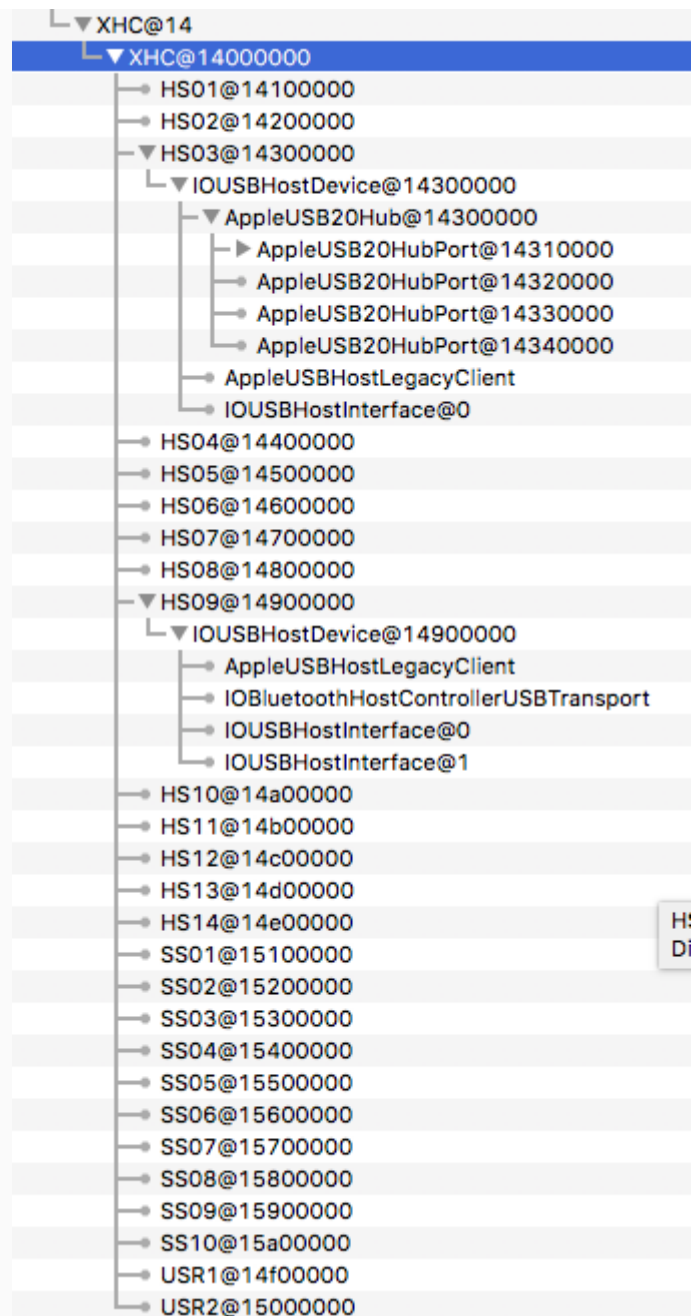
Test each port by inserting a USB2 device, then remove it, then insert a USB3 device and remove it. If you look carefully while you insert and remove the devices you can see which ports are assigned to which physical port (you will see the changes in ioreg).

Note: If you have a USB3 hub it can make it easier to test each port. Since a USB3 hub connects to both the USB3 pins and the USB2 pins of a USB3 port (and can be used in a USB2-only port), and it requires no action to eject properly, you will find it much quicker to use a USB3 hub instead of a set of USB2/USB3 memory sticks. In the examples that follow, I'm using a USB3 hub to test ports.

DO NOT USE the 'Search' box in IORegistryExplorer.app! Just look at the correct section of the ioreg tree on the left pane (scroll as necessary to find the EH01/EH02/XHC as applicable to your hardware).

Note that to test each port properly, you may need to move any USB keyboard or mouse device around so that you can test each port with both USB2 and USB3 (or USB3 hub).

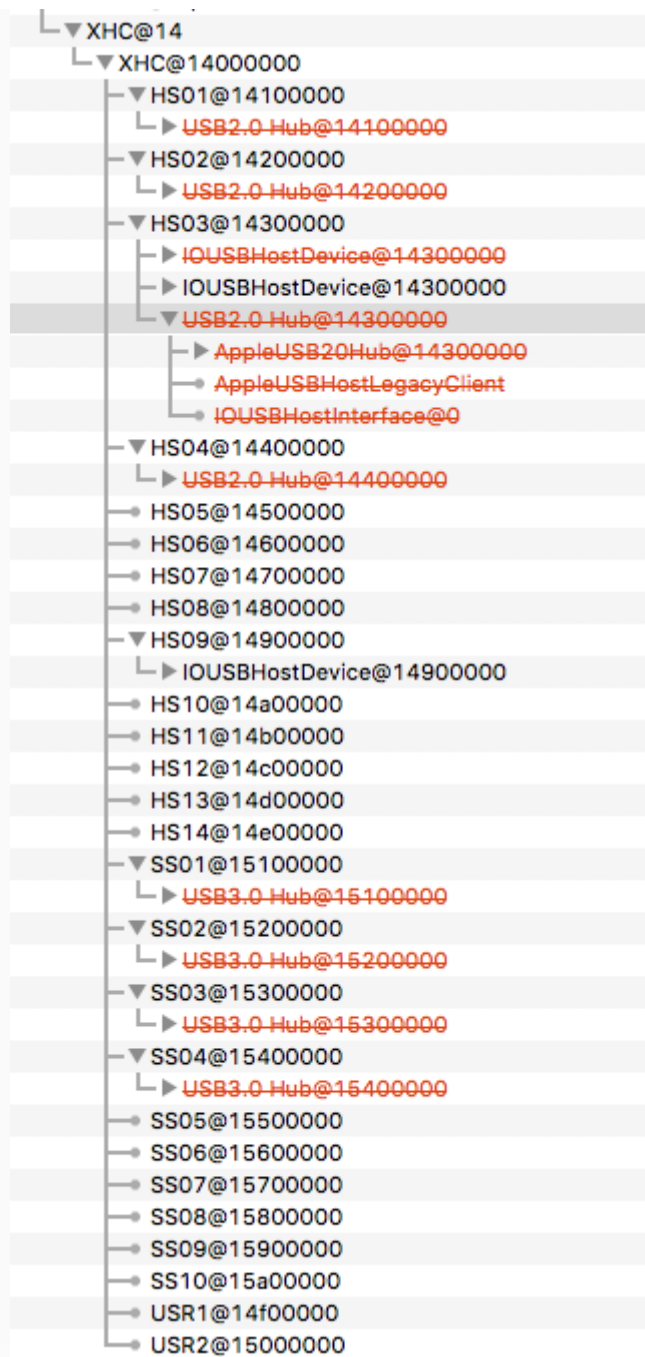
On my NUC6i7KYK, this is XHC before starting to test the USB ports:



You can see that only the HS03 port is being used. The HS03 port happens to be where my Dell U3011 USB hub is plugged in, which also happens to have my USB keyboard/mouse dongle. The HS09 is the internal bluetooth controller.

After testing all ports:





I usually monitor ioreg as I plug and unplug devices so I know where the ports are located physically.

As you can see, the following ports are used:

HS01/SS01: USB3 front left

HS02/SS02: USB3 front right

HS03/SS03: USB3 rear bottom

HS04/SS04: USB3 rear top

HS09: bluetooth

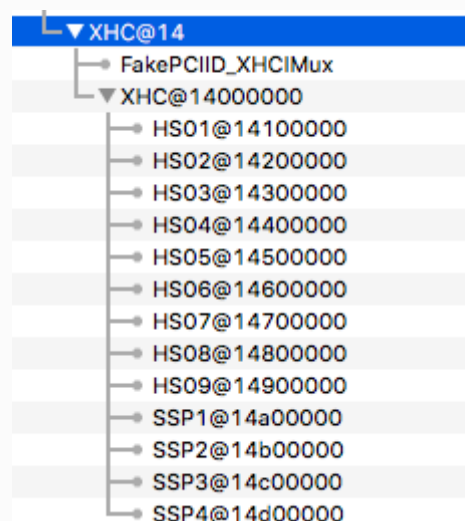
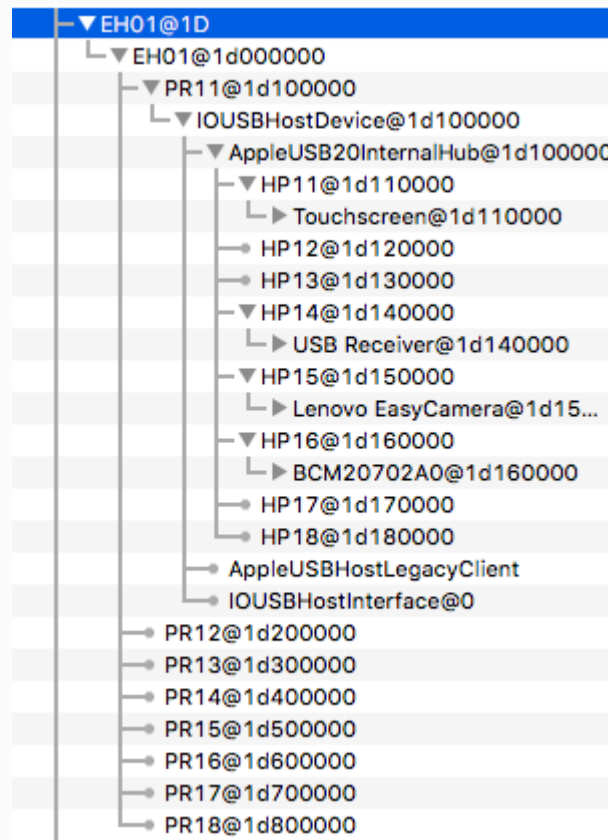
Note that the SSxx ports will have a corresponding HSxx port. The SSxx port (or SSPx) is used when you plug in a USB3 device. The HSxx port is used when you plug in a USB2 device. USB2 only ports will use only HSxx.



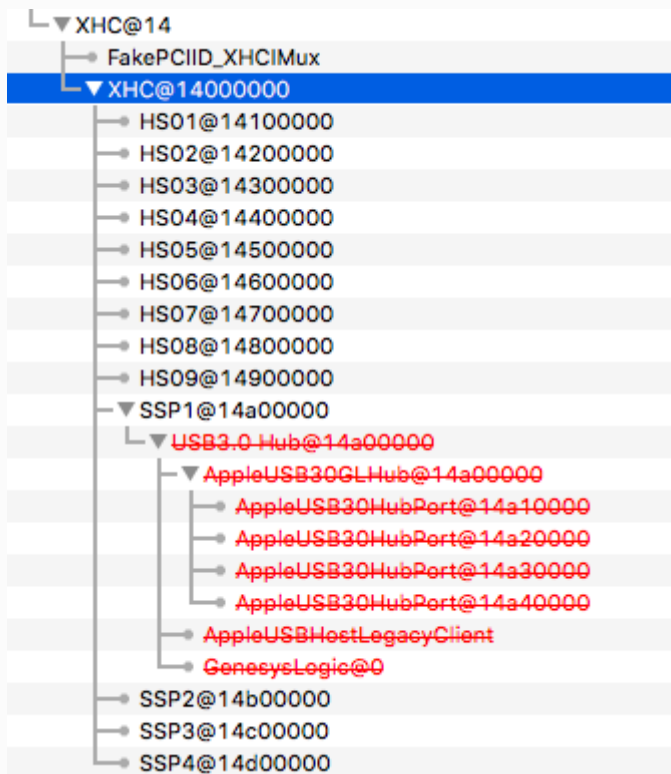
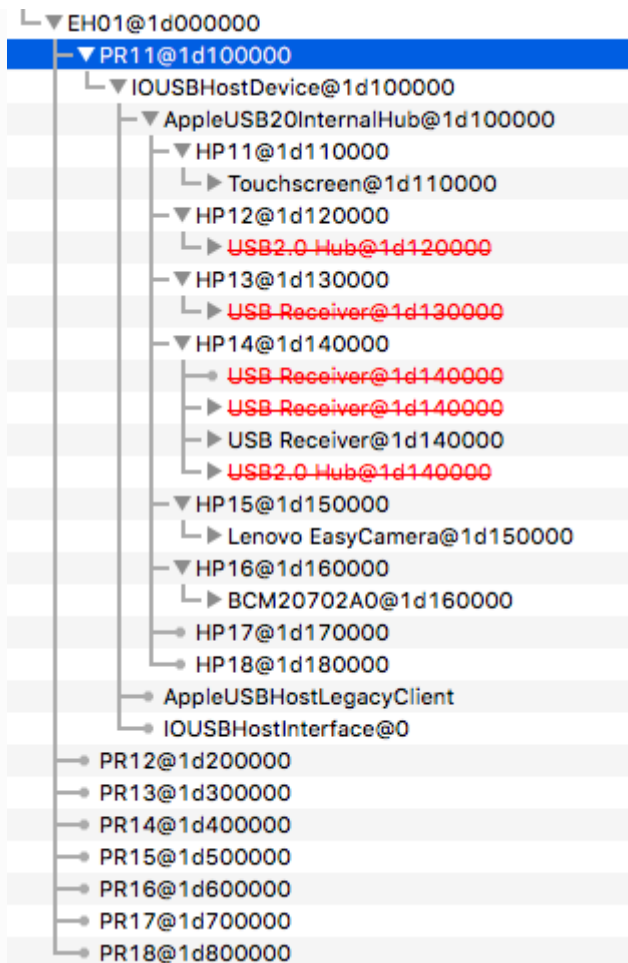
So, although the controller used on this Skylake computer has the potential for 26 ports (HS01-HS14, SS01-SS10, USB1, USB2), only 9 ports are connected. We can eliminate HS05-HS08, HS10-HS14, SS05-SS10, USB1, and USB2.

Let's take a look at the test results for my Lenovo u430 (using FakePCIID\_XHCIMux.kext).

Before testing:



After testing all ports:



Ports used (EH01/HUB1/XHC):

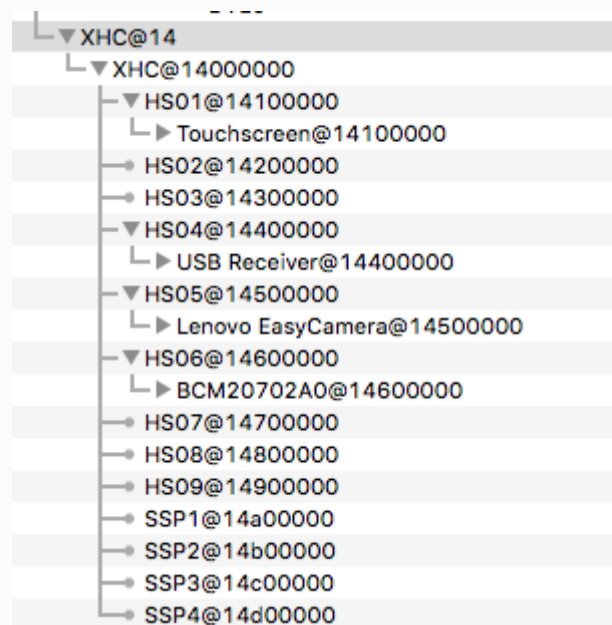
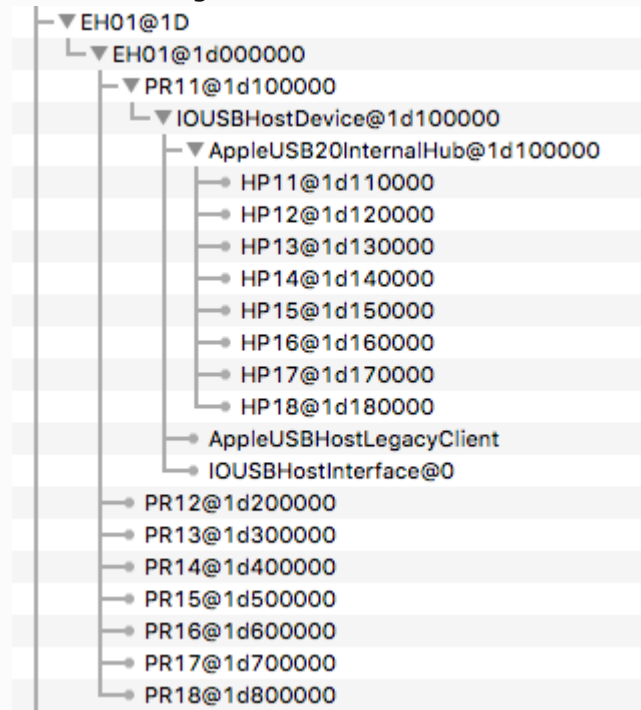
PR11: internal hub

HP11: touchscreen

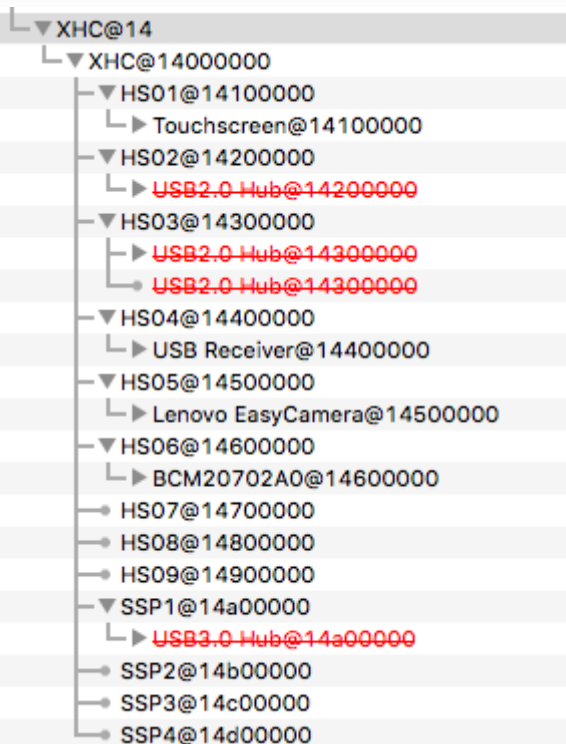
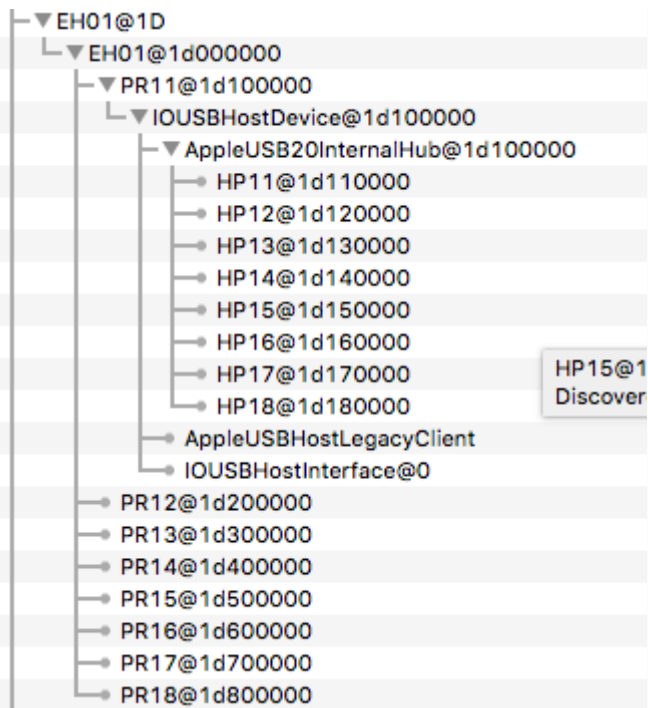
HP12/SSP1: USB3 left  
HP14: USB2 far right  
HP15: USB2 near right  
HP15: camera  
HP16: bluetooth

Or, if we remove FakePCIID\_XHCIMux.kext:

Before testing:



After testing:



Ports used (only on XHC):

HS01: touchscreen

HS02/SSP1: USB3 left

HS02: USB2 far right

HS03: USB2 near right

HS05: camera

HS06: bluetooth

As you can see, FakePCIID\_XHCIMux.kext has the effect of moving all HSxx ports to EHCI (to the internal hub on

EH01.PR11).

After you have collected the data in IORegistryExplorer, and you are certain that is accurate, you are now prepared to create a custom SSDT based on the data you have obtained in ioreg.

### **Creating custom SSDT for USBInjectAll.kext**

USBInjectAll.kext is coded in a data driven way. Nothing is hard-coded in the kext code itself. All data is contained in the Info.plist. When USBInjectAll.kext starts, matching on EH01/EH02/XHC, or the internal hubs connected to EH01/EH02 port 1, it consults the configuration data in Info.plist as related to the device it has attached to and injects the data it finds there. But an SSDT can be used to override the data in the Info.plist.

A template is provided in the USBInjectAll.kext github repo, SSDT-UIAC-ALL.dsl, which has overrides for all the data already in the USBInjectAll.kext Info.plist. You could compile SSDT-UIAC-ALL.dsl to SSDT-UIAC-ALL.aml (File Save As, format: ACPI Machine Language Binary in MaciASL) and place it in ACPI/patched, and although all the data in the Info.plist will be overridden by the SSDT, no net change would be observed, as the data in SSDT-UIAC-ALL.dsl is the same as the data already in the USBInjectAll.kext Info.plist.

In order to effect our changes, we must modify the SSDT-UIAC-ALL.dsl so it contains only the ports we need.

The steps are as follows:

- use SSDT-UIAC-ALL.dsl as a template (<https://github.com/RehabMan/OS-X-USB-Inject-All/raw/master/SSDT-UIAC-ALL.dsl>)
- remove configuration sections that don't apply to the target hardware
- remove ports from the various sections that are not needed
- change UsbConnector values to match physical hardware/ports

The first step is to eliminate configuration data from SSDT-UIAC-ALL.dsl that don't apply to the target hardware and USB configuration.

For example, if no ports are active on EH01/EH02 or the related USB hubs (on PR11/PR21), you can eliminate the configuration data for EH01/EH02/HUB1/HUB2. In the case of Skylake, it has

no EHCI controllers, so EH01/EH02/HUB1/HUB2 configuration can automatically be removed. Same would be true if you're disabling EHCI either via BIOS or ACPI (covered later). And you can eliminate any XHC configuration that does not match your XHC device-id. Look in ioreg to find your XHC device-id.

For example, you can see that the NUC6i7KYK uses 0xa12f:

**XHC@14**

Class Inheritance: IOPCIDevice : IOService : IORegistryEntry : OSObject

Bundle Identifier: com.apple.iokit.IOPCIFamily

- SBUS@1F,4
  - XHC@14
    - XHC@14000000
  - PDRRC
  - PRRE
  - PTID
  - PTMD
  - PWRB
    - AppleACPIButton
      - IOHIDEventServiceUserClient
  - RMNE@0
  - RTC
    - AppleRTC
  - SIO1

Property	Type	Value
built-in	Data	<00>
class-code	Data	<30 03 0c 00>
compatible	Data	<"pci8086,2064", "pci8086,a12 class,0c0330", "XHC">
device-id	Data	<2f a1 00 00>
IODeviceMemory	Array	1 value
IOInterruptControllers	Array	2 values
IOInterruptSpecifiers	Array	2 values
IOName	String	pci8086,a12f
IOPCIMSIMode	Boolean	True
IOPCIResourced	Boolean	True

So, for the NUC's XHC, it will match against the data for "8086\_a12f" in the SSDT-UIAC-ALL.dsl. We can eliminate all other XHC configuration data.

For the NUC6i7KYK, we end up with this starter template:

Code (Text):

```
// Initial trimmed SSDT-UIAC.dsl for NUC6i7KYK
DefinitionBlock ("", "SSDT", 2, "hack", "UIAC", 0)
{
    Device (UIAC)
    {
        Name (_HID, "UIA00000")
        Name (RMCF, Package ()
        {
            "8086_a12f", Package ()
            {
                "port-count", Buffer () { 26, 0, 0, 0 },
                "ports", Package ()
                {
                    "HS01", Package ()
                    {
                        "UsbConnector", 3,
                        "port", Buffer () { 1, 0, 0, 0 },
                    },
                    "HS02", Package ()
                    {
                        "UsbConnector", 3,
                        "port", Buffer () { 2, 0, 0, 0 },
                    },
                    "HS03", Package ()
                    {
                        "UsbConnector", 3,
                        "port", Buffer () { 3, 0, 0, 0 },
                    },
                    "HS04", Package ()
                    {

```

In the case of the NUC6i7KYK, it matches against a section in SSDT-UIAC-ALL.dsl that matches specifically against "8086\_a12f". Some XHC devices will match against only part of the device-id. For example, the u430 device-id is 0x9c31. As there is no specific configuration for "8086\_9c31", it will match against "8086\_9xxx". Always use the template data that has the most specific match for your device-id. For example, if your device-id is 0x9cb1, you should use the template data for "8086\_9cb1", not "8086\_9xxx".

After you trim the template down to include only the configuration sections you need, now it is time to further trim the data to eliminate unused ports. As you recall, the NUC6i7KYK uses only HS01-HS04, HS09, and SS01-SS04.

The resulting SSDT-UIAC.dsl for NUC6i7KYK with only the ports we need:

Code (Text):

```
// Port trimmed SSDT-UIAC.dsl for NUC6i7KYK
DefinitionBlock ("", "SSDT", 2, "hack", "UIAC", 0)
{
    Device(UIAC)
    {
        Name(_HID, "UIA00000")
        Name(RMCF, Package()
        {
            "8086_a12f", Package()
            {
                "port-count", Buffer() { 26, 0, 0, 0 },
                "ports", Package()
                {
                    "HS01", Package()
                    {
                        "UsbConnector", 3,
                        "port", Buffer() { 1, 0, 0, 0 },
                    },
                    "HS02", Package()
                    {
                        "UsbConnector", 3,
                        "port", Buffer() { 2, 0, 0, 0 },
                    },
                    "HS03", Package()
                    {
                        "UsbConnector", 3,
                        "port", Buffer() { 3, 0, 0, 0 },
                    },
                    "HS04", Package()
                    {
                        "UsbConnector", 3,
```

We can comment each port based on notes we took about each:



**Code (Text):**

```
// Port trimmed with port comments SSDT-UIAC.dsl for NUC6
DefinitionBlock ("", "SSDT", 2, "hack", "UIAC", 0)
{
    Device(UIAC)
    {
        Name(_HID, "UIA00000")
        Name(RMCF, Package()
        {
            "8086_a12f", Package()
            {
                "port-count", Buffer() { 26, 0, 0, 0 },
                "ports", Package()
                {
                    "HS01", Package() // HS USB3 front le
                    {
                        "UsbConnector", 3,
                        "port", Buffer() { 1, 0, 0, 0 },
                    },
                    "HS02", Package() // HS USB3 front ri
                    {
                        "UsbConnector", 3,
                        "port", Buffer() { 2, 0, 0, 0 },
                    },
                    "HS03", Package() // HS USB3 rear bot
                    {
                        "UsbConnector", 3,
                        "port", Buffer() { 3, 0, 0, 0 },
                    },
                    "HS04", Package() // HS USB3 rear top
                    {
                        "UsbConnector", 3,
```

And we should change the UsbConnector values to match the type of port. The defaults are not always correct (USBInjectAll has no way to know what the correct value is).

Common port connector types are USB2 = 0, USB3 = 3, internal = 255.

USB type C ports can be either 9 or 10, which depends on how the hardware deals with the two possible orientations of a USB type C device/cable.

If a USB-C uses the same SSxx in both orientations, then it has an internal switch (UsbConnector=9).

If a USB-C uses a different SSxx in each orientation, then it has no switch (UsbConnector=10).

HSxx ports that are connected to a USB3 port should be marked UsbConnector=3, not UsbConnector=0.

Note: Read the ACPI spec for more valid values (\_UPC).

In the case of the NUC6i7KYK, only HS09 needs adjusting. Since the bluetooth controller is internal, it should be 255.

The final SSDT-UIAC.dsl is:

Code (Text):

```
// Port trimmed with port comments and correct UsbConnect
DefinitionBlock ("", "SSDT", 2, "hack", "UIAC", 0)
{
    Device (UIAC)
    {
        Name (_HID, "UIA00000")
        Name (RMCF, Package ()
        {
            "8086_a12f", Package ()
            {
                "port-count", Buffer () { 26, 0, 0, 0 },
                "ports", Package ()
                {
                    "HS01", Package () // HS USB3 front le
                    {
                        "UsbConnector", 3,
                        "port", Buffer () { 1, 0, 0, 0 },
                    },
                    "HS02", Package () // HS USB3 front ri
                    {
                        "UsbConnector", 3,
                        "port", Buffer () { 2, 0, 0, 0 },
                    },
                    "HS03", Package () // HS USB3 rear bot
                    {
                        "UsbConnector", 3,
                        "port", Buffer () { 3, 0, 0, 0 },
                    },
                    "HS04", Package () // HS USB3 rear top
                    {
                        "UsbConnector", 3,
```

Now we are ready to compile this file and place it in EFI/Clover/ACPI/patched. In MaciASL, you can compile as AML by using File -> Save As, format: ACPI Machine Language Binary. Save it to a location you know how to navigate to (desktop). Then mount your EFI partition and copy it to EFI/Clover/ACPI/patched. I suggest using SSDT-UIAC.aml for the name.

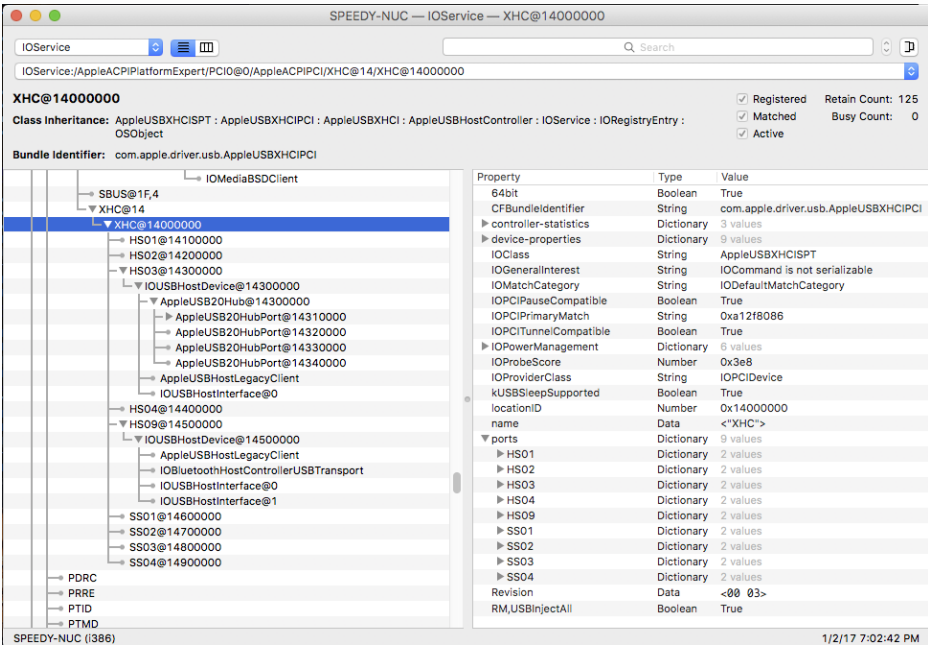
Note: If you're using config.plist/ACPI/SortedOrder, you must insure that the new SSDT is listed in SortedOrder. If SortedOrder is specified, Clover will load only those SSDTs listed in SortedOrder.

After you have copied the SSDT to ACPI/patched, you are ready to test. If it does not work (ports missing, port discovery done

incorrectly, wrong template used, etc), you can always use -uia\_ignore\_rmcf to disable the override code in SSDT, and then check your work.

Once your SSDT is working, you can remove or disable the port limit patch.

As you can see from the image below from the NUCi7KYK, we now have just the ports that we need injected and are well under the 15-port limit:



### Lenovo u430 example with EH01 and HUB1

Since Skylake doesn't have an EHCI controller, we only needed to customize for XHC. The Lenovo u430 uses an 8-series chipset, therefore it has an EHCI controller. Although it can be disabled (which I actually do for my u430 in real use), this walk through will keep it enabled as a way of demonstrating EHCI and the related hub configuration.

Just as we did with the Skylake NUC6, we start by removing all configuration sections not needed from SSDT-UIAC-ALL for the Lenovo u430 based on the data we collected with FakePCIID\_XHCIMux.kext. The XHC controller is 0x9c31, so we need only "8086\_9xxx" for the XHC configuration, and we need EH01 and HUB1 for the ports on the EHCI controller and related hub.

The resulting trimmed SSDT is:

Code (Text):

```
// Initial trim SSDT-UIAC.dsl for u430
DefinitionBlock ("", "SSDT", 2, "hack", "UIAC", 0)
{
    Device(UIAC)
    {
        Name(_HID, "UIA00000")

        Name(RMCF, Package()
        {
            "HUB1", Package()
            {
                "port-count", Buffer() { 8, 0, 0, 0 },
                "ports", Package()
                {
                    "HP11", Package()
                    {
                        //"UsbConnector", 0,
                        "portType", 2,
                        "port", Buffer() { 1, 0, 0, 0 },
                    },
                    "HP12", Package()
                    {
                        //"UsbConnector", 0,
                        "portType", 2,
                        "port", Buffer() { 2, 0, 0, 0 },
                    },
                    "HP13", Package()
                    {
                        //"UsbConnector", 0,
                        "portType", 2,
                        "port", Buffer() { 3, 0, 0, 0 },
                    }
                }
            }
        }
    }
}
```

If we installed that SSDT unmodified at this point, because all possible ports are included, we would have same result as without it.

Also note it is good practice for the XHC configuration to change to the actual device id, in this case "8086\_9c31". In this case, either will work "8086\_9xxx" or "8086\_9c31", but using a specific configuration identifier helps identify the specific XHC device this SSDT was made for.

Note: You can also use "XHC".

So, after removing ports that are not used, and changing the XHC configuration identifier to "8086\_9c31":

Code (Text):

```
// Port trimmed SSDT-UIAC.dsl for u430
DefinitionBlock ("", "SSDT", 2, "hack", "UIAC", 0)
{
    Device(UIAC)
    {
```

```

Name(_HID, "UIA00000")
Name(RMCF, Package()
{
    "HUB1", Package()
    {
        "port-count", Buffer() { 8, 0, 0, 0 },
        "ports", Package()
        {
            "HP11", Package()
            {
                //"UsbConnector", 0,
                "portType", 2,
                "port", Buffer() { 1, 0, 0, 0 },
            },
            "HP12", Package()
            {
                //"UsbConnector", 0,
                "portType", 2,
                "port", Buffer() { 2, 0, 0, 0 },
            },
            "HP13", Package()
            {
                //"UsbConnector", 0,
                "portType", 2,
                "port", Buffer() { 3, 0, 0, 0 },
            },
        },
    },
}

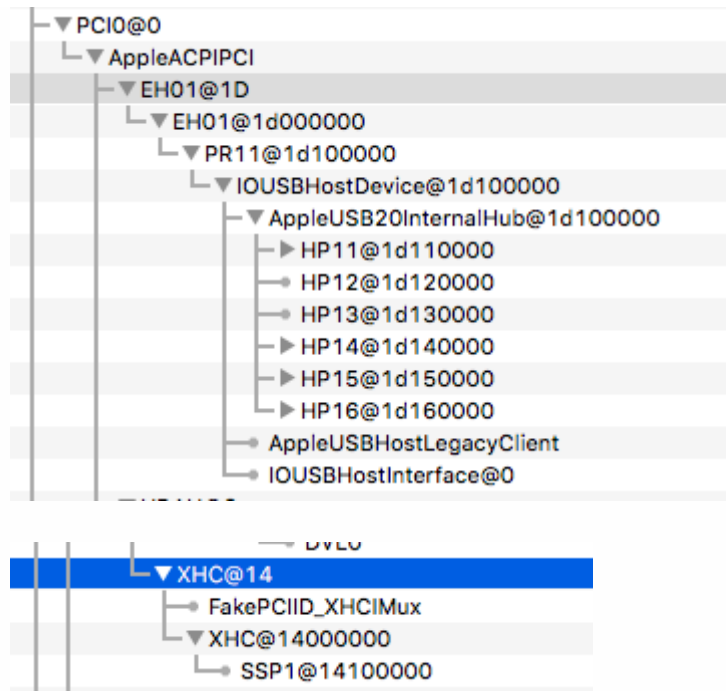
```

In this case, there is no need to modify any of the UsbConnector values, since they are already set correctly. The USB Hub driver in macOS/OS X does not use UsbConnector and we don't really understand the purpose of portType. You can experiment with different values if you need to. Refer to the hub driver Info.plist in IOUSBHostFamily.kext/Contents/PlugIns/AppleUSBHub.kext/Contents/Resources for ideas.

Note: If anyone has a Sierra or El Capitan ioreg from a MacPro6,1, MacBookPro8,1, MacBookPro6,1, iMac13,x, or MacBookPro8,3 along with information about the physical hub ports where portType=2 or portType=0 is used, please share.

After we install this SSDT, remove -uia\_ignore\_rmcf, we can verify in ioreg and test that all ports work.

Here we have images of the ports injected on EH01, HUB1, and XHC:



Note that for my Lenovo u430, I decided to disable the EHCI controller and have everything on XHC. FakePCIID\_XHCIMux.kext is not installed.

The resulting SSDT is as follows (in this SSDT, I used "XHC" instead of "8086\_9xxx" or "8086\_9c31" just to show it is possible):

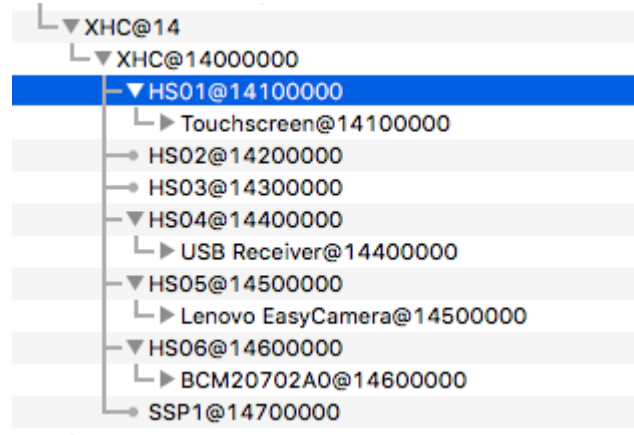
Code (Text):

```
// With EHCI disabled, SSDT-UIAC.dsl for u430
DefinitionBlock ("", "SSDT", 2, "hack", "UIAC", 0)
{
    Device (UIAC)
    {
        Name (_HID, "UIA00000")
        Name (RMCF, Package ()
        {
            "XHC", Package ()
            {
                //"port-count", Buffer () { 0x0d, 0, 0, 0 }
                "ports", Package ()
                {
                    "HS01", Package () // touchscreen
                    {
                        "UsbConnector", 255,
                        "port", Buffer () { 0x01, 0, 0, 0 }
                    },
                    "HS02", Package () // HS USB3 left
                    {
                        "UsbConnector", 3,
                        "port", Buffer () { 0x02, 0, 0, 0 }
                    },
                    "HS03", Package () // USB2 far right
                    {
```

```
"UsbConnector", 0,  
    "port", Buffer() { 0x03, 0, 0, 0  
},  
    "HS04", Package() // USB2 near right  
{  
    "UsbConnector", 0
```

Don't expect to understand the code under "Disabling EHCI #1" unless you have had a good read of the ACPI specification and the Intel 8-series chipset datasheet.

Here is the result with everything on XHC (no EH01 to be found in ioreg):



## Problem Reporting

If you have a problem, please describe the problem clearly, make sure your profile accurately describes your hardware and provide all the data as requested in the FAQ.

Read FAQ, "Problem Reporting"

<https://www.tonymacx86.com/threads/faq-read-first-laptop-frequent-questions.164990/>

**Compress all files as ZIP.** Do not use external links. Attach all files using site attachments only.

Pabloesc, Marovincian, brengos and 29 others like this.

Last edited: Sep 17, 2018