Kext as Kext can oder USB 3.0 ohne USBInjectAll

Beitrag von „Brumbaer“ vom 16. August 2017, 15:07

Worum geht's hier eigentlich ?
Das Folgende beschreibt eine Möglichkeit USB 3.0 Anschlüsse ohne USBInjectall zum Laufen zu bekommen.
Der Sinn dieses Artikels ist allerdings nicht einen Ersatz für USBInjectAll zu schaffen, sondern Hintergrundwissen über Kexte zu vermitteln.
Für eine vollständige Beschreibung der Funktionsweise, Anwendung und Einsatz von Kexten seien Interessierte an die Apple Dokumentation verwiesen.
Das beschriebene Verfahren lässt sich auf alle Arten von Treibern anwenden, ob Bluetooth, WiFi oder Grafik.
Das hier gegebene Beispiel wurde unter High Sierra getestet und sollte genauso unter Sierra funktionieren.

Kext, ein Kext ?
Kexte erlaube es einem Programme als wären sie Teil des Kernels ablaufen zu lassen.
Man kann drei generelle Arten von Kexten unterscheiden, die die eine Routine enthalten die einfach gestartet wird, dann Services, die mit Treibern vergleichbar sind und Bibliotheken, die anderen Kexten Routinen zur Verfügung stellen.
Uns interessieren besonders die Kexte die Treiberfunktion haben.

Jeder hat sein Bündel zu tragen
Ein Kext ist ein Bundle - ein Bündel. Ein Bundle tritt nach aussen hin wie eine Datei auf, ist aber ein Ordner. Beim Doppelclick im Finder öffnet sich der Ordner nicht, sondern macht je nach Art des Bundles etwas anderes.
Apps sind Bundles und was die bei einem Doppelclick machen, weiß jeder.
Ein Doppelclick auf ein Kext führt zur Nachfrage, was man denn mit dem Ding machen will, wie bei einem unbekannten Dateityp.
Allerdings kann man doch in ein Bundle hineinsehen. Im Finder macht man einen Rechtsclick auf das Bundle und über den Menüpunkt "Paketinhalt zeigen" kann man das Bundle wie einen Ordner öffnen und seinen Inhalt sehen und bearbeiten.

Es endet mit einem .kext
Kexte sind Bundles also Ordner die mit der Endung .kext enden.
Wie wir schon festgestellt haben, kann man ein Kext nicht mit einem Doppelclick starten.

Stattdessen werden Kexte beim Systemstart geladen, wenn sie in einem der folgenden Ordner liegen:

- /System/Library/Extensions
- /Library/Extensions

Es genügt aber nicht, dass das Kext in einem dieser Ordner liegt, es muss auch

- als Eigentümer "root"
- als Gruppe "wheel"
- und die Rechte 755 haben.

Wenn man ein Kext mit einem Hilfsprogramm wie Kext Utility installiert werden die drei Dinge automatisch angepasst. Kopiert man das Kext händisch muss man selbst Sorge dafür tragen, dass sie angepasst werden.

Verwendet man Clover kann man Kexte beim Systemstart laden, indem man sie in den EFI/CLOVER/kexts/Other Ordner auf der EFI-Partition legt.
Clover sind Eigentümer, Gruppe und Rechte egal, deshalb genügt das Kopieren des Kextes in den Ordner.

Form ohne Inhalt ist nichts.
Wie gesagt ein Kext ist ein Bundle ist ein Ordner.
Der Inhalt eines Kextes ist in gewissem Rahmen vorgeschrieben.
Der eigentliche Inhalt kann direkt im Kext(dem Ordner) liegen. Häufig befindet sich aber im Kext ein Unterordner Contents, der dann den eigentlichen Inhalt enthält.
Ob man einen Contents Ordner zwischen schaltet oder nicht, bleibt jedem selbst überlassen.
Beide Varianten sind gleichwertig.

- Das Kext muss eine Datei Info.plist enthalten.
- Wenn das Kext ausführbaren Code enthält liegt er im Unter-Ordner MacOS.
- Wenn ein Kext Unter-Kexte hat, liegen diese im Unter-Ordner PlugIns.
- Wenn ein Kext signiert ist, liegt die zugehörige Datei im UnterOrdner _CodeSignature. Über SIP kann man die Signatur Überprüfung abschalten. um nicht signierte oder Kexte mit falscher Signatur (weil man was am Kext geändert hat).

Legt man die Sachen in anderen Ordnern ab, findet macos sie einfach nicht.
Das Kext kann weitere Dateien und Ordner enthalten.

Ein schöner Vertreter seiner Art

Schauen wir uns doch mal ein Kext an.

IOUSBHostFamily verwendet einen Contents Ordner.
In dem gibt es die oben angeführten Dinge plus *version.plist*.
Die *version.plist* hat zusätzliche Versionsinformationen und ist für die Funktion nicht interessant.
Alleine anhand der Namen kann man schon ein paar Rückschlüsse ziehen

- **IO**: Ein- Ausgabe
- **USB**: betrifft USB
- **Host**: keine Endgeräte, sondern die, die den Bus kontrollieren.
- **Familiy**: Gilt für eine ganze Reihe von Geräten.

Die allgemeine Funktionalität wird von IOUSBHostFamily zur Verfügung gestellt, aber es gibt PlugIns, die die Details für bestimmte Controller oder Controller Arten abhandeln.
Auch bei den Plugins helfen die Namen weiter.
AppleUSB*HCIPCI sind

- **Apple** eigene Treiber
- für **USB** Controller
- der Typen **OHCI**, **UHCI**, **XHCI**, **EHCI**,
- die am **PCI** Bus angeschlossen sind.

Ein stolzer Spross der Familie
Wir interessieren uns für USB 3.0 d.h. XHCI Controller. Die Controller sind über PCI angeschlossen.
Schauen wir uns also das *AppleUSBXHCIPCI.kext* an.

Es zeigt sich ein inzwischen vertrautes Bild:

_CodeSignature enthält die Signatur, kümmert uns nicht.
MacOS enthält die Datei mit der Software, da reinzuschauen und zu analysieren was passiert, führt viel zu weit. Uns genügt es zu wissen, dass, da die Datei AppleUSBXHCIPCI in MacOS liegt, das Kext eigenen Programmcode enthält.
version.plist ist wie gesagt nur Versionsinformation
Und das lässt nur noch die *Info.plist* übrig.

Informationen sind heutzutage Alles

Die *Info.plist* ist an jedem Kext das Interessanteste.
Eine plist ist eine Datei, die eine Liste von Eigenschaften enthält, eine Property List halt. PLists haben ein festes Format, so wie Word Files auch.
Eine PList ist ein XML Dokument und es gibt auch eine DTD dazu.
Man kann eine PList mit einem Text Editor oder einem XML Editor bearbeiten.
XCode enthält auch einen PList Editor, der das Bearbeiten etwas komfortabler als mit einem Texteditor macht.
XCode ist jedem zugänglich, also verwende ich im Folgenden dessen PList Editor, denn seine Darstellung ist übersichtlicher als die im Texteditor und er kümmert sich um die Datenstruktur, So werden Fehler bei Änderungen vermieden.

Die Darstellung ist vergleichbar mit der Listendarstellung im Finder. Statt Dateien haben wir Einträge der Datentypen Bool, Data, Date, Number oder String. Statt Ordner haben wir Container in Form von Dictionary oder Array.

Wie man Einträge ändert, löscht und hinzufügt, bitte ich, falls es sih einem nicht von selbst erschließt, im Netz zu recherchieren.

So sieht die *Info.plist* von *AppleUSBXHCIPCI.kext* aus

IOKitPersonalities ist mit Absicht nicht aufgeklappt, denn es ist ellenlang und ich möchte erst über die anderen Dinge sprechen.
Viele Einträge dienen nur der Information des Lesers, aber einige sind für die Funktion essentiell.

Wenn ein Kext geladen wird, startet es nicht automatisch. Es wird erstmal im Kernel mit einem Namen versehen abgelegt.
Diesen Namen findet man unter "*Bundle identifier*". Es kommt vor, dass man sich auf ein anderes Kext bezieht, dann braucht man diesen Namen.
Wir haben gesehen, dass dieses Kext Programmcode enthält (Datei im MacOS Ordner). Der Dateiname wird unter "*Executable file*" eingetragen. Macos sucht im *MacOs* Ordner des Kextes nach der Datei,
"*OSBundleRequired*" gibt an wozu das Kext gebraucht wird. Das ist in den meisten Fällen "*Root* ". Kexte für Netzwerkfunktionen haben oft "*Network-Root*" und Kexte, die auch im abgesicherten Modus geladen werden sollen "*Safe Boot*".
Das Kext kann Bibliotheken benötigen. Welche es benötigt und deren Version steht unter "*OSBundleLibraries*". Sollte das System keine kompatiblen Bibliotheken finden, startet das Kext nicht.

Starke Persönlichkeiten oder gespaltene Persönlichkeit ?
Und jetzt wird es spannend *IOKitPersonalities*.
Wenn es diesen Eintrag gibt, liegt ein Kext mit Service (Treiber) Funktion vor.

Gibt es einen solchen Eintrag, so enthält er einen oder mehrere Einträge für Services, die gestartet werden können.
Jeder Eintrag enthält Bedingungen, die erfüllt sein müssen damit der Service gestartet wird. Dabei sind die Services voneinander unabhängig, es sein denn man macht einen explizit von einem anderen abhängig.
Dieser Überprüfung findet zu bestimmten Zeitpunkten bzw. bei bestimmten Ereignissen statt. D.h. auch wenn ein Service nicht gleich gestartet wird, so kann er noch später, sobald alle Bedingungen erfüllt sind, gestartet werden.

Schauen wir uns solch einen Eintrag an:

Von diesen Einträgen sind zwei Startbedingungen.
Die erste ist "*IOProviderClass*". Dort wird der Name eines Services angegeben. Wird solch ein Service gestartet, dann ist es das Startsignal, dass die Überprüfung losgeht.
In diesem Fall heißt der Service *IOPCIDevice*. Für jedes PCI Gerät wird solch ein Service gestartet. Damit ist schon mal klar, dass sich der Service um den es hier geht um ein PCI Gerät kümmert.

Welches Schweinderl hätten sie denn gerne ?
Aber welches ?
"*IOPCIClassMatch*" gibt an, welcher Geräteklasse, das Gerät angehören muss, damit der Service in Frage kommt. In diesem Falle 0x0c033000. In der PCIe Spezifikation sind die Klassen und ihre Werte beschrieben.

- 0C Serielles Gerät
- 03 USB
- 30 XHCI

In diesem Fall gibt es keine weiteren Bedingungen.
Der Service ist also zuständig für alle PCI Geräte mit der Klasse XHCI Controller.
Der Programmcode des Service steht im Bundle "*CFBundleIdentifier*" in diesem Falle "*com.apple.driver.usb.AppleUSBXHCIPCI*".
Ein kurzer Blick nach oben zeigt und, dass das das Kext selbst ist.
Der Programmcode kann mehr als einen Service enthalten. "*IOClass*" gibt an welcher

verwendet werden soll. In diesem Fall "*AppleUSBXHCIPCI*".
"*IOPCIPauseCompatible*" und "*IOPCITunnelCompatible*" haben mit dem Teilen von PCI Kanälen bzw. dem Anschluss über Thunderbolt zu tun und interessieren uns hier nicht.

Eins geht noch
Ein zweites Beispiel

Die Überprüfung beginnt natürlich ebenfalls mit dem Finden eines PCI Gerätes "*IOProviderClass*".
Doch diesmal haben wir kein *IOPCIClassMatch*. Statt dessen haben wir ein *IOPCIPrimaryMatch*.
PCI Geräte haben eine Primary und eine Secondary Id.
Sie bestehen jeweils aus einer Produkt- und einer Hersteller-Id. Beides sind 16bit Hex Werte, die Produkt Id kommt zuerst.
Also der Service kommt in Frage, wenn ein Gerät mit der Produkt-Id 0x9c31 und der Hersteller-Id 0x8086 (steht für Intel) gefunden wird.
Der ProgrammCode befindet sich in diesem Kext und heißt *AppleUSBXHCILPT*.
Man kann davon ausgehen dass auch dieses Gerät der PCI Klasse XHCI Controller angehört.

Es kann nur einen geben
Woher weiß das System nun welchen Treiber es nehmen soll, diesen oder den den wir uns vorher angeschaut haben ?
"*IOProbeScore*" hilft an der Stelle. Es wird der Treiber mit dem höheren *IOProbeScore* genommen.
Wird also ein Device mit der Primary-Id 0x9c318086 gefunden so hat der Service *AppleUSBXHCILPT* ein *IOProbeScore* von 1000, *AppleUSBXHCI* einen *IOProbeScore* von 0 (kein Eintrag bedeutet 0).
Also wird in dem Fall *AppleUSBXHCILPT* verwendet. Bei XHCI Geräten mit anderen Primary Ids kommt der Eintrag nicht in Frage, aber *AppleUSBXHCI* ist für diese immer noch ein Kandidat.

Soviel hierzu
Und schon haben wir die erste Gruppe von IOKitPersonalities analysiert

All diese Einträge stellen Treiber für verschiedene XHCI Controller zur Verfügung.
Es gibt Treiber für bestimmte Controller und einen generischen Treiber für die Controller, für die es keinen besonderen Eintrag gibt.

Alle Treiber basieren auf dem *AppleUSBXHCI* Treiber, sie sind Subklassen von *AppleUSBXHCI* dem generischen Treiber.

Wenn wir nun ein Board mit einem macos nicht bekannten XHCI Controller verwenden würden, würde macos den generischen Treiber verwenden, solange der Controller zur PCI-XHCI-Klasse gehört - wovon auszugehen ist.
Das ist ne tolle Sache, wenn der generische Treiber mit dem Controller funktioniert.
Nun kann es aber sein, dass es nicht der Fall ist. In diesem Fall könnte man einen neuen Eintrag für diesen Controller anlegen, als IOPrimaryMatch dessen Id eintragen und dann als IOClass alle von den anderen verwendeten Treiber durchprobieren bis man einen findet der funktioniert.
Später dazu mehr.

Einen hab ich noch
In den IOKitPersonalities dieses Kextes gibt es noch eine zweite Art von Einträgen:

Anhand der Namen kann man darauf schliessen, dass sie dazu dienen Anpassungen in Abhängigkeit vom vorliegenden Rechner durchzuführen.
In diesem Fall dienen sie dazu dem Treiber mitzuteilen, welche Ports des Controllers bei diesem Computer verwendet werden.

Wann geht's los ?
"*IOProviderClass*" zeigt uns "*AppleUSBXHCIPCI*". D.h. wird ein Service dieses Namens oder eine Subklasse davon geladen, dann startet die Überprüfung.
Wir müssen nicht mal raten, wann dies der Fall ist. Wir wissen, dass *AppleUSBXHCIPCI* oder eine seiner Subklassen gestartet wird, wenn ein Treiber für einen XHCI Controller gefunden wird.

Und sonst ?
"*IONameMatch*" ist der Name dec PCI Gerätes. Damit der Test erfolgreich ist muss der Name "*XHC1*" sein.
Das genügt noch nicht das "*model*" muss vom Typ "*iMac17,1*" sein.
Also dieser Service wird gestartet, wenn ein XHCI Treiber geladen wurde, das PCIGerät *XHC1* heißt und der Rechner ein *iMac17,1* ist.
Der Service der gestartet wird befindet sich nicht in diesem Kext, denn der "*CFBundleIdentifier* " ist "*com.apple.driver.AppleUSBMergeNub*". Wo das Kext tatsächlich steht ist egal, denn alle Kexte wurden ja schon geladen und mit ihrem Bundle-identifier gespeichert.

Der Service heißt laut "*IOClass*" Eintrag "*AppleUSBMergeNub*".
Merge bedeutet zusammenführen. Da werden wohl Eigenschaften zusammengeführt. D.h. dem USB Treiber (XHCI Controller sind USB Controller) werden Parameter übergeben.
Der Name "*IOProviderMergeProperties*" ist ein deutlicher Hinweis, dass es sich hier um die Parameter handelt.

Wie man sieht, sieht man bei Data nichts. Das liegt daran, dass der PList Editor versucht die Daten als Text anzuzeigen, es aber gar kein Text ist.
Ein Rechtsclick auf das Fenster und Show Raw Keys/Values angewählt ändert das

Das kenn ich doch
Wer sich mit USB und DSDT/SSDT beschäftigt hat, dem kommt das sehr vertraut vor.
port-count ist die höchste verwendete Port Id.
Die Ports werden meist einfach durchnummeriert die HS Ports beginnen bei 1, die SS Ports bei 17. 17 macht Sinn, wenn man Hexadezimal denkt, denn dann ist es 0x11.
Wenn 0x1a, siehe *port-count*, die höchste verwendete Port Id ist, müssten wir einen SSP10 Eintrag haben. Haben wir aber nicht. Vermutlich hat der Programmierer einfach den höchsten bei diesem Controller möglichen Wert genommen - so ein Faultier.

HS02 ist der Name unter dem das Port verwaltet wird. Typischerweise HS für High Speed und SS oder SSP für Super Speed Port.
Unter "*USB Connector*" wird die Anschlussart angegeben. Damit ist der Stecker bzw. die Buchse gemeint mit dem/der der Port verbunden ist.

- 0 ist die normale USB2.0 Typ A Buchse - die breite Flache in Schwarz. Nur HS.
- 3 ist die normale USB3.0 Typ A Buchse - die breite Flache in Blau. SS und HS.
- 9 ist USB-C mit HS und SS.
- 255 ist ein unbekannter oder eigener Stecker. MoBo Header fallen in diese Rubrik.

HS02 liegt also an einer USB3.0 Buchse und die Port Id ist 0x02. Das ist der zweite SS Port am Controller - deshalb auch HS02.

Das Kext für EHCI is genauso organisiert. Dort gibt es keinen Eintrag für iMac17,1, denn der iMac17,1 hat nur einen XHCI Controller und keinen EHCI Controller.

Wird ein PCI Gerät erkannt, checkt das Kext ob es sich um einen XHCI Controller handelt und

wenn dem so ist, lädt es einen Treiber.
Wenn es danach einen passenden Eintrag für Rechner und Gerätenamen findet, wählt es die dazu passenden Ports aus und teilt sie dem Treiber mit, findet es keinen Eintrag, verwendet der Treiber die ersten 15 Ports des Controllers.

Das Problem bei einem Hack ist, dass die Ports meist nicht die sind, die man gerne hätte. Sollte man mit einem Board ein SMBIOS verwenden, das eine Controllerart die das Board hat nicht verwendet, werden die ersten 15 Ports davon eingetragen, was u.U. nicht hilfreich ist.

Und was hilft mir das ?
Das sehen wir morgen im nächsten Beitrag.