Ozmosis BIOS Guide – individuelles Anpassen & Erklärung des Aufbaus

Beitrag von „kuckkuck“ vom 6. Oktober 2017, 20:35

In diesem Post will ich jedem Ozmosis Nutzer das individualisieren des eigenen Ozmosis UEFIs ein wenig näher bringen und evtl. ein wenig die Augen öffnen, was das bearbeiten von Ozmosis ROMs angeht. Dabei werde ich darauf eingehen, wie ein jeder die Kern-Komponenten von Ozmosis identifizieren kann, den Inhalt von ROMs kürzen und verstehen kann und eigene Dateien hinzufügen kann... Vielleicht hilft das ja dem ein oder anderen Interessenten die Ozmosis Welt ein wenig besser zu verstehen 👍

Als erstes muss ein jeder Ozmosis Nutzer verstehen, dass auch das Ozmosis Paket für jeden PC angepasst werden muss, um perfekt zu funktionieren.
Dazu kann unter anderem zählen:

- Anpassen und individualisieren der **Defaults.plist**
- Installieren der **richtigen Kexts**
- Patchen von Kexts mit dem **KernextPatcher** (**KextToPatch** (**KernextPatcher**) sowie **ACPIPatcher**)
- Einrichten von **DarBoot** (ab APFS)
- Anpassen von ACPI Tabellen wie das **patchen der DSDT** und **erstellen von SSDTs**
- Auswählen des gewünschten **Themes**
- Hinzufügen von **UEFI Applikationen** wie der **HermitShell**

und so weiter...

Jede der oben aufgezählten Möglichkeiten kann dabei über die EFI gelöst werden (Wie das geht ist jeweils verlinkt). Sobald man sich das ganze jedoch ein bisschen genauer anschaut, erkennt man, dass all diese Dinge auch über das ROM passieren können.

Hierfür ein Beispiel warum dies eventuell sinnvoll ist:
Erstellt man eine individuelle Defaults.plist und setzt diese in die EFI in den Ordner Efi/Oz, wird diese nach einem NVRAM Reset genutzt. Aber woher stammen die SMBIOS Werte bevor man seine eigene Defaults.plist einsetzt?
Diese kommen aus einer Defaults.plist, die sich im BIOS befindet und in das installierte

Ozmosis BIOS eingefügt wurde. Beim Einstellen einer Defaults in der EFI Partition, wird die im ROM befindliche Defaults.plist lediglich überschrieben/ersetzt und ist von da an im Ruhestand.

Bevor man jetzt alles doppelt moppelt, kann man nun also einfach, wenn man will, seine eigene Defaults ins ROM einsetzen und sie danach wieder aus der EFI entfernen.

Gleiches gilt auch für Kexts: Kopiert man eine aktuelle Version des FakeSMC nach Efi/Oz/Darwin/Extensions/Common, ersetzt diese lediglich die im ROM vorhandene FakeSMC oder SMCEmulator.kext.

Ebenfalls kann es für manche sinnvoll sein auf ein vollständiges BIOS zu setzen. Bei einem ROM mit zB. integrierter Ethernet Kext, kann schon während der Installation Ethernet für Anmeldungen, iCloud o.ä genutzt werden. Der Hackintosh ist komplett funktionstüchtig, selbst wenn auf einer neuen Platte installiert wird und dementsprechend keinerlei Dateien auf der EFI Partition liegen.

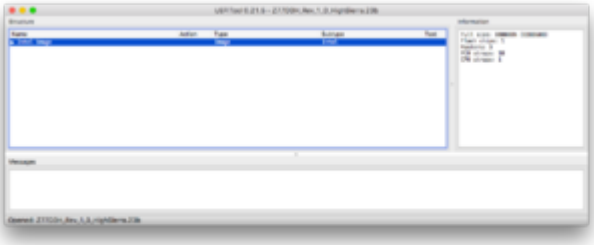
Das macht es vielleicht jetzt erstmal logisch sein ROM auf jeden Fall anzupassen, aber lasst mich euch warnen:
Das Kopieren aller existierenden Dateien aus dem EFI in das ROM, bringt keinen Vorteil was Performance oder ähnliches angeht! Im Gegenteil, das Einfügen möglichst vieler KernelExtensions in das BIOS, verlangsamt sogar den Boot Prozess!
Trotzdem bin ich ein Freund von Struktur und habe in meinem ROM gerne auch wirklich nur das was ich brauche. Dabei geht es vorallem um folgendes:

- Entfernen von ungenutzten Ozmosis Dateien und Kexts aus dem BIOS, um die Firmware nicht unnötig voll zu pressen
- Einfügen der für den eigenen Bedarf benötigten Tools. Dabei will der eine unbedingt eine Shell im BIOS, und der andere braucht einen KernextPatcher um Kexts und/oder Kernel zu patchen
- Ersetzen von Standard Konfigurationen mit der eigenen Konfiguration: Löschen der standardisierten Defaults.plist, einfügen der eigenen Defaults.plist

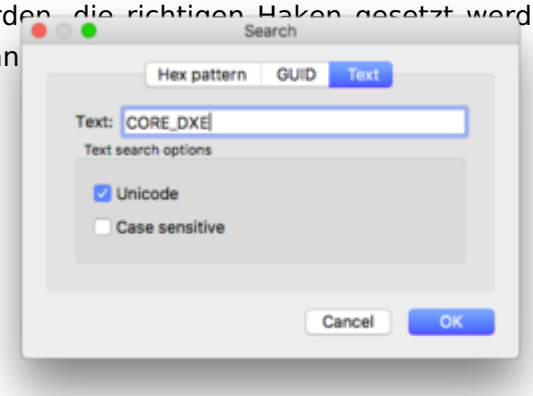
Aber fangen wir doch einfach mal an!
Ich werde euch in diesem ersten Guide nicht zeigen, wie ihr euer eigenes BIOS von Vanilla zu einem Ozmosis BIOS macht, das ist ein anderes Thema (Wer genau aufpasst sollte aber nach den Guides fähig sein, sein eigenes ROM aus einem original BIOS zu schaffen). Deswegen nehmen wir als Basis einfach die vorgefertigten ROMs aus der Downloadsektion... Jedes BIOS sieht anders aus, aber alle haben bestimmte Komponenten für Ozmosis enthalten, je nach

Platz, mehr oder weniger. In diesem Fall werde ich ein Z77 BIOS verwenden, da in diesen ROMs viel Platz und dementsprechend auch die größte Anzahl an Ozm-Komponenten zu finden ist. Das ganze lässt sich jedoch danach auf andere ROMs übertragen...

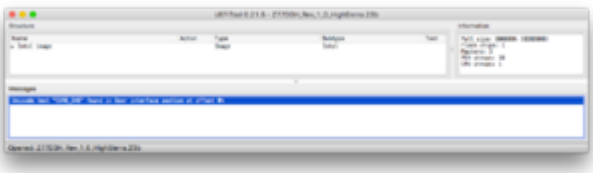
Für unsere Aufgabe eignet sich am aller besten das UEFI-Tool, da dieses das ROM in einer sehr übersichtlichen GUI darstellt und uns automatisch vor Fehlern bewahrt.
UEFI-Tool installiert und gestartet müssen wir einfach nur unser Ozm BIOS aus der Datenbank nehmen und auf das UEFI Fenster per Drag und Drop ziehen. Alternativ geht dies auch über Menüleiste-->File-->Open Image File (All files).
Ist dies passiert wird uns der Inhalt des ROMs präsentiert. Das kann dann zB so aussehen:



Das ganze ist eine Ordnerstruktur wie man sie vom Finder kennt. Es gibt mehrere Firmware Volumen für unterschiedliche Zwecke mit unterschiedlichen Größen, aber das soll uns alles garnicht interessieren, denn uns interessiert nur alles was Ozmosis betrifft! Beim anfertigen eines Ozmosis ROMs, wird lediglich am richtigen Ort im ROM ein wenig Platz geschaffen und daraufhin verschiedenste Dateien, zuständig für den erfolgreichen Boot von OS X eingefügt. Diese Datein kommen immer an die gleiche Stelle und auch nur diese Stelle interessiert uns. Dabei handelt es sich um das Volumen in dem die Datei namens `CORE_DXE` zu finden ist. Bevor wir jetzt aber das durchkämmen anfangen, suchen wir einfach danach.
Mit CMD/Win+F öffnet sich ein Suchfenster. Hier muss jetzt die Spalte "Text" ausgewählt werden, die richtigen Haken gesetzt werden und nach CORE_DXE gesucht werden. Das sieht dann



Bestätigen wir das ganze mit OK, erhalten wir



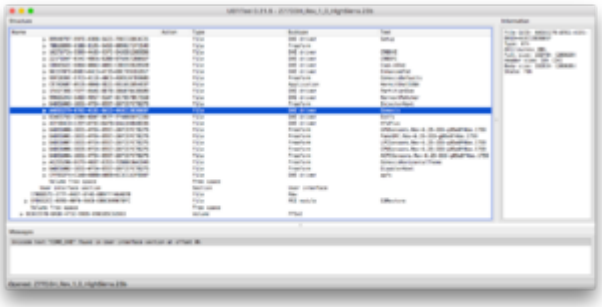
einen Eintrag in "Messages":

Ein einfacher Doppelklick auf den Eintrag bringt uns auch direkt an die gewünschte Stelle, und das sieht so aus:

Jeder einzelne Eintrag ist ein Firmware-Modul, das sich im BIOS befindet. Die Einträge tragen:

- eine GUID: die lange Nummer, jede Nummer kommt nur einmal im ganzen ROM vor
- eine Art des Verzeichnis (Type): Region, Volume, File, Section, FreeSpace etc.
- eine bestimmte Dateiart (Subtype): Wie zB Treiber (DXEDriver), Programme wie die Shell (Application), undefinierte Freiformen (Freeform) etc.
- und meistens einen DateiName (Text): Dieser text sagt meistens um was es sich genau handelt und ist das am einfachsten identifizierbare Merkmal

Soeben haben wir nach dem Text `CORE_DXE` gesucht und landen dementsprechend bei einer Datei mit dem Namen CORE_DXE. Im gleichen Volumen wie diese Datei, befinden sich alle für Ozmosis relevanten Dateien. Häufig sind diese dabei am Ende des Abschnitts zu finden. Wenn wir also runterscrollen werden wir meistens direkt mit einigen Dateien konfrontiert, die je nach ROM unterschiedlich sind, **eine von ihnen mit dem Namen `Ozmosis`**
So sieht das ganze aus in meinem Beispiel:



In meinem Beispiel gehören hier ziemlich viele Dateien zu Ozmosis (von `Enhanced Fat` bis `apfs`), die ich im folgenden aufzählen und kurz beschreiben werde:

- `Ozmosis`: Dies ist der KernTreiber in dem das Hexenwerk passiert und der für das meiste zuständig ist. Dieser Treiber braucht manchmal Updates, genauso wie Clover updates braucht, um mit neuen macOS Versionen zu funktionieren. Diese Updates sind aber **viel** seltener als bei Clover! **In jedem BIOS enthalten!**

- `OzmosisDefaults`: Ganz einfach: Eine Defaults.plist, konvertiert in eine Version die ins BIOS kann! Bei den meisten wird es sich hier um eine standardisierte Version handeln, die in den ROMs aus unserer DownloadSektion untergebracht wird. **In jedem BIOS enthalten!**
- `EnhancedFat`: Ein Treiber, der das Benutzen von FAT Dateisystemen ermöglicht. So kann an erster Stelle unsere EFI gelesen und von Ozmosis verarbeitet werden. **In jedem BIOS enthalten!**
- `HFSPlus`: Ebenfalls ein FileSystem-Treiber, der das Verwenden von HFS Platten ermöglicht. HFS wird von Apple schon lange genutzt und wurde nun von APFS und HighSierra ersetzt. HighSierra kann jedoch nach wie vor mit HFS benutzt werden, weshalb dieser Treiber auch für HighSierra sehr wichtig ist. **In jedem BIOS vor Mojave enthalten!**
- `apfs`: Der Apple FileSystem DateiSystem-Treiber. Dieser ermöglicht das Benutzen von APFS Platten und muss in jedem High Sierra ROM enthalten sein.
  Alternativ kann der ApfsDriverLoader genutzt werden (am besten mit Kext2FFS in ffs umwandeln) dieser ist wesentlich kleiner und lädt die zum OS passende APFS Version. **In jedem BIOS ab High Sierra enthalten!**
- `FakeSMC`, `SMCEmulator` oder `VirtualSMC`: Dies ist eine weit bekannte KernelExtension die den Apple SMC simuliert. Entweder FakeSMC **oder** SMCEmulator **oder** VirtualSMC muss in jedem BIOS enthalten sein. **In jedem BIOS enthalten!**

All diese Dateien sind also in jedem ROM zu finden! Sie sind die Minimalkonfiguration eines mit Ozm ausgestatteten BIOS!

Alle folgenden Dateien sind "freiwillig" und werden je nach Platz im ROM mit untergebracht.

- `OzmosisHorizontalTheme`: Das Theme, auch bekannt als GUI oder UserInterface, das man sich beim Start des PCs als Bootauswahl anzeigen lassen kann. **Für Multiboot-Systeme stark empfohlen.**
- `DarBoot`: Ein kleines Programm von Cecekpawon, dass dafür sorgt, dass alle macOS Booteinträge angezeigt werden und auch APFS Booteinträge bei einem NVRam-Reset nicht verloren gehen. **Empfohlen für alle Systeme ab und einschließlich High Sierra.**
- `KernextPatcher`: Cecekpawon's Kext und Kernel Patcher, der Kernel- und KextsToPatch Einträge über eine KernextPatcher.plist in /Efi annehmen kann.
  Der KernextPatcher ist in einer speziellen Form in jedem Mojave-kompatiblen ROM enthalten!
- `AcpiPatcher`: Cecekpawon's ACPI und DSDT Patcher, der ACPI-Renames über eine AcpiPatcher.plist in /EFI annehmen kann und ebenfalls CPU-SpeedStep-States generieren kann.
- `ExtFS`: Das Ext Dateisystem, das viel bei Linux genutzt wird. Wer kein Linux nutzt, braucht dies im Normalfall auch nicht.

- `HermitShellX64`: Eine Variante des UEFI Shell Programms, über das Änderungen am System gemacht werden können. Das Programm kann unter anderem auf den NVRAM und die Dateien eines Systems zugreifen, wodurch zB. Veränderungen an der EFI vorgenommen werden können. Häufig erscheint diese als Eintrag "Built-In Shell" in der Bootauswahl des Mainboards.
- `GPU-, LPC-, ACPI-, CPUSensors`: Die FakeSMCSensoren, die häufig gerne im BIOS untergebracht werden. Mit ihnen lässt sich unter anderem die Hardware überwachen/überprüfen. Dabei können zB mit dem OS X Programm "HWMonitor", Temperaturen, Taktraten und Fan-Geschwindigkeiten ausgelesen werden.
- `PartitionDXE`: Treiber, der für die Unterstützung von ungewöhnlichen Partitionstabellen wie Apple Partition Map oder hybrid GPT/MBR sorgt. Dieser Treiber ist überwiegend für SnowLeopard Installationen und nicht für neuere macOS notwendig!
- `InjectorKext`: Kernel Extension, notwendig für den Boot von SnowLeopard. NEIN! Diese Kext ist heutzutage nicht für das Injecten von Kexts zuständig! Man braucht sie lediglich für SnowLeo.
- `DisablerKext`: Kext die das Power Management für Mac OSX SnowLeopard deaktiviert. Auch dieser Kext ist für neuere OS X nicht notwendig!
- `VoodooHDA`: Veraltete Kext für AudioLösungen. Diese ist in den meisten ROMs nicht mehr zu finden, da mittlerweile überwiegend auf andere Lösungen gesetzt wird.
- `Andere Kexts`: *Prinzipiell lässt sich jede Kext ins ROM integrieren. Dementsprechend ist es auch möglich das man hier zB EthernetKexts wie Realtek oder IntelMausi findet, jedoch auch USBInjectAll und alles sonst noch denkbare.*

Ihr seht, es gibt sehr viele Möglichkeiten zu einem BIOS, von der minimal Konfiguration abweichende Dateien hinzuzufügen... Umso schwerer ist es also für die Ersteller der ROMs einen Mittelweg zu finden, um jeden User happy zu machen. Das ganze geht aber nicht immer so, und während der eine das Theme vermisst, hat jemand anderes ein ungenutztes Theme und vermisst die Shell.
Deswegen will ich euch zeigen wie ihr euer BIOS anpassen könnt um euren Ansprüchen gerecht zu werden und nebenbei die Funktionsweise der einzelnen Komponenten erklären, damit dem ein oder anderen ein paar Hintergründe klar werden. Vielleicht weckt die Sache ja auch so euer Interesse, dass ihr gerne selber für andere Ozmosis fähige ROMs bauen wollen würdet, denn das ganze ist eigentlich wirklich kein Hexenwerk.

~~So ich habe jetzt erstmal genug geschrieben und brauche eine Pause. Der Guide wie ihr das ROM modifiziert folgt in Kürze!~~

Und hier gehts weiter: Ozmosis BIOS Ausmisten und Erweitern

Beitrag von „kuckkuck“ vom 6. Oktober 2017, 21:08

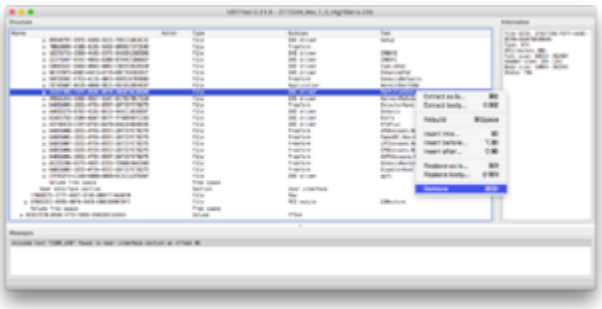
Das Ausmisten

Da ihr ja jetzt wisst wozu welche Komponente zuständig ist, können wir uns ans ausmisten machen.
Das ganze geht super einfach mit ein paar Klicks. Zuerst müssen wir uns aber im klaren sein, was wir brauchen und was wir wollen.

Machen wir mal das Beispiel mit SnowLeopard: Muss auf diesem Hacky Snowleopard laufen/wird es jemals benutzt werden?
Wenn deine Antwort "nein" ist, können wir alle Komponenten die für SnowLeo integriert wurden schon einmal entfernen:
`PartitionDXE`, `InjectorKext`, und `DisablerKext`

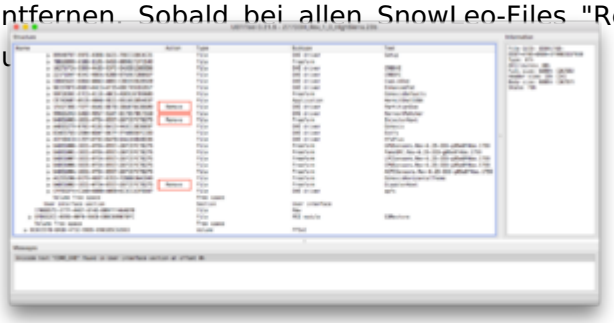
Das ab jetzt beschriebene Prozedere ist identisch für alles andere das entfernt werden soll. Wichtig ist nur, dass alle zur Ozmosis Minimalkonfiguration zählenden Komponenten nicht entfernt werden!
Wer also ExtFS, Shell, Kexts (wie das meist unbenutzte VoodooHDA) oder anderes entfernen will, kann dies mit diesem Guide ebenfalls tun.

Mit UEFI Tool können Komponenten mit einem Rechtsklick --> "Remove" entfernt werden. Das sieht dann so aus:



Klicken wir jetzt auf "Remove", wird die Datei noch nicht sofort aus dem ROM entfernt. Dies passiert immer erst sobald wir speichern! Das gibt uns jedoch wiederum die Möglichkeit direkt mehrere Komponenten auszuwählen, und alle SnowLeopard Dateien in einem Rutsch zu

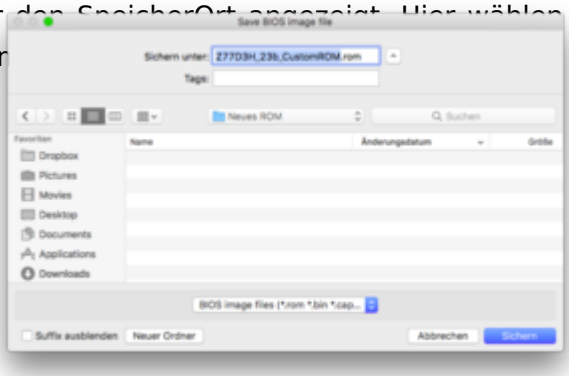
entfernen. Sobald bei allen SnowLeo-Files "Remove" ausgewählt wurde, sollte das ganze so au



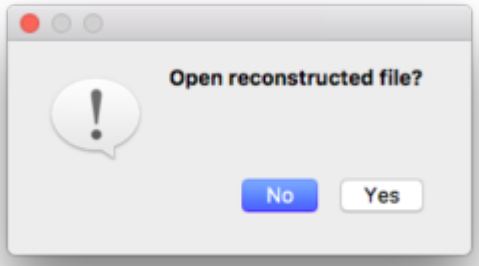
Man beachte dabei die kleinen Einträge in der

Spalte Action, die in diesem Fall "Remove" heißen (Rot markiert)
Zeit für unseren ersten Test, um zu sehen ob das auch so klappt.

Wir drücken also CMD+S oder gehen auf File --> "Save image file..." und uns wird ein Fenster für den SpeicherOrt angezeigt. Hier wählen wir einen Ordner und Dateinamen aus, was dann zur



Klicken wir jetzt auf "Sichern", erscheint nach



kurzer Zeit folgende Meldung:

Hier immer "Yes" auswählen, denn nur so können wir überprüfen ob alles richtig gelaufen ist!

Sobald das frisch gebackene ROM geöffnet ist, müssen wir auf die Sektion "Messages", am unteren Ende des UEFI Tools schauen. Stehen hier **keine Meldungen** hat alles gut funktioniert und unser ROM ist OK. In diesem Zustand könnte es jetzt direkt geflasht werden! Tauchen jedoch Meldungen auf, ist dies kein gutes Zeichen und es ist etwas schief gelaufen. Falls diese Meldungen nach einem weiteren Versuch wieder auftauchen, oder sie bereits im ROM aus der Datenbank erscheinen, könnte es sich um unwichtige Meldungen handeln. Habt ihr Fragen dazu, könnt ihr einfach eure Meldung in diesen Thread schreiben.

Aber wir sind noch nicht fertig, denn da geht noch einiges mehr!

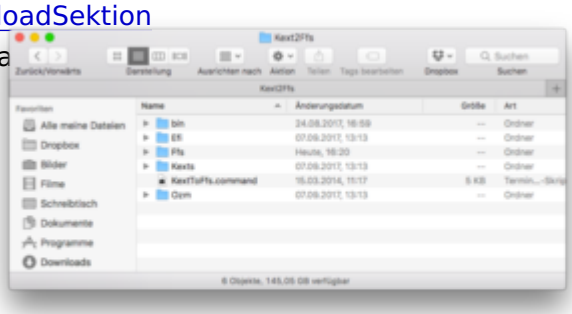
Das Hinzufügen

Durch das Entfernen von ungenutzten Komponenten, kann Platz im ROM für neues geschaffen werden. Manche ROMs haben von Haus aus genug Platz um noch weiteres aufzunehmen, trotzdem lohnt es sich immer komplett ungenutzte Dateien aus dem ROM zu entfernen.

Praktisch an UEFITool ist, dass wir automatisch davor beschützt werden bestimmte Fehler zu machen. So kann zB nicht zu viel in das ROM eingefügt werden. Versuchen wir eine Datei hinzuzufügen, die den Speicherplatz sprengt, wird uns das Tool beim Speichern eine Fehlermeldung ausgeben und das ROM nicht speichern lassen. Wir brauchen uns also nicht darum kümmern, dass das BIOS zu voll wird, denn wenn wir etwas hinzufügen wollen, was zu groß ist, werden wir das schon merken. Dazu aber später mehr.

Wie kann man jetzt Dateien zum ROM hinzufügen?
Dies geht nur mit Dateien die im Flash-File-System - Format vorliegen, kurz `.ffs`
Eine Defaults im Plist Format, lässt sich also nicht ohne weiteres zum ROM hinzufügen...

Um Dateien zu FFS zu konvertieren, wird häufig das Tool "Kext2FFS" benutzt, welches ihr hier herunterladen könnt: DownloadSektion
Heruntergeladen und entpa



einfachen Finder Ordner handelt, der so aussieht:

Jeder Unterordner hat eine eigene Funktion:
In `bin` sind alle Executables untergebracht, die das Kext2FFS Skript verwendet. Dieser Ordner ist für uns relativ uninteressant.

In `EFI`, `Kexts` und `Ozm` können wir unsere Dateien hinterlegen, die zu FFS umgewandelt werden sollen.

In `EFI` können .efi Dateien gelegt werden. Dazu zählen zB FileSystem-Treiber wie APFS.efi, UEFI Applications wie Shell.efi oder auch DXE-Treiber wie Ozmosis.efi selber. Bei Problemen nach der Umwandlung und anschließenden Integration ins Rom, lohnt sich manchmal die Nutzung folgender Version: Kext2FFS
Diesen Ordner braucht Otto NormalVerbraucher nicht häufig, da die meisten eigentlichen .efi Dateien, auch bereits als .ffs gefunden werden können, so zB Ozmosis oder der HFSPlus, die man häufig direkt als Ozmosis.ffs oder HFSPlus.ffs finden kann.

Weiter gehts mit dem Ordner `Kexts`. Wie der Name schon sagt, können wir hier jede beliebige Kernel-Extension zu FFS konvertieren lassen, um sie danach ins BIOS zu integrieren.
Hier aber nochmal die Warnung: In das BIOS sollten nur die nötigsten Kexts eingefügt werden, da sie das Hochfahren verlangsamen können. Was sich gut anbietet, ist mit diesem Tool die neueste FakeSMC sowie die zugehörigen Plugins nach .ffs zu konvertieren, um diese ins BIOS einzufügen. Wer will kann sich auch einen Lan Treiber ins BIOS packen, um Lan bei Neuinstallationen zu haben, man beachte aber, dass dies wie oben erwähnt nicht immer die beste Idee ist.

Schließlich gibt es noch den Ordner `Ozm` mit dem sich bei dieser Kext2FFS-Version .plist Dateien nach FFS umwandeln lassen. (Der Ordner kann in manchen Fällen auch "OzmDefault" lassen und neben dem Ordner "Ozm" stehen. In diesem Fall einfach "Ozm" ignorieren und "OzmDefault" nutzen.)
Dieser Ordner wird in erster Linie dazu benutzt um eine Defaults.plist nach FFS umzuwandeln. Falls man seine eigene Defaults in sein ROM integrieren will, muss diese in oben genannten Ordner gelegt werden und das Ende (Suffix) .plist besitzen.

Jetzt gibt es nur noch 2 Dinge in Kext2FFS: Der `Ffs` Ordner und der `KextToFfs.command`:
Der `KextToFfs.command` ist das Skript, dass alle unsere hinterlegten Dateien nach .ffs umwandelt, sobald wir es ausführen. Das Skript läuft dabei im Terminal und gibt uns noch ein paar Informationen aus.

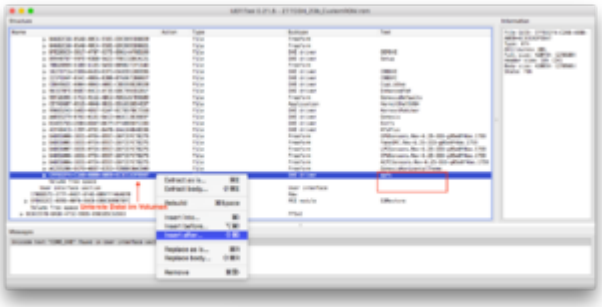
Sobald der Befehl dann ausgeführt wurde (doppelklick), erscheinen in dem Ordner `Ffs` in dem jeweiligen Unterordner `Efi`, `Kext` oder `Ozm` unsere erstellten FFS Dateien, jeweils einmal in normal, und einmal in einer komprimierten, kleineren Form, im UnterOrdner `Compress`. Die Dateien werden dabei nach dem OriginalDatei-Namen benannt. Heißt unsere Original-Datei

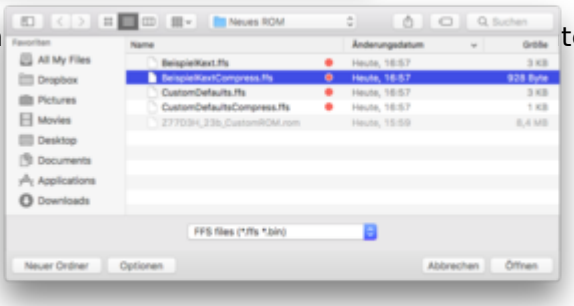
also "Spongebob", wird die erstellte Datei ebenfalls "Spongebob" heißen, und das auch im UEFITool und boot.log!

Haben wir jetzt also unsere Dateien, die eingebaut werden sollen, im FFS Format, können wir uns auch schon ans einbinden machen. In den meisten Fällen spricht nichts dagegen einfach die `Compress`-Variante einzufügen, da diese häufig wesentlich kleiner ist.

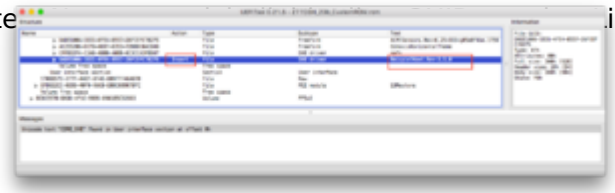
In meinem Fall werde ich nun eine Custom- Defaults.plist und eine BeispielKext einfügen. Dafür haben ich diese mit Kext2FFS konvertiert und die .ffs Dateien in meinen Ordner verschoben.

Sobald UEFI-Tool geöffnet ist, gilt wieder das Standardprozedere: BIOS-File auf das ProgrammFenster ziehen, nach `CORE_DXE` suchen und zu den Ozmosis Komponenten scrollen.
Hier angekommen müssen wir jetzt nur noch die Dateien einfügen. Dafür wählen wir die unterste Ozmosis-Datei im ausgewählten Volume aus, machen einen Rechtsklick und wählen "Insert after...":

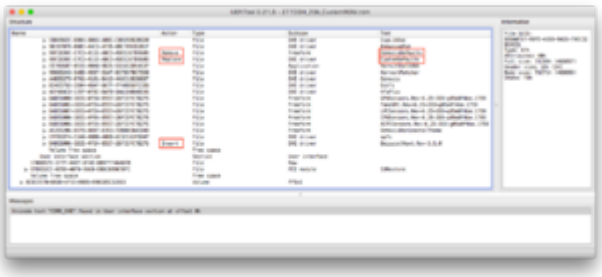


Im erscheinenden 

teien in der Compress-Version und klicken öffnen:

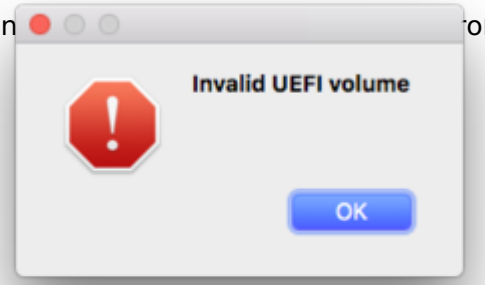
Dadurch wird die Datei mit unserem gewählte  iste hinzugefügt und die Action "Insert" hinterlegt:

Bevor wir speichern, ersetzen wir jetzt noch die Standard `OzmosisDefaults`, mit unserer Defaults.plist, in meinem Fall CustomDefaults.
Dazu suchen wir die Datei mit dem Namen "OzmosisDefaults", machen einen Rechtsklick und wählen jetzt "**Replace as is...**"!
Im erscheinenden Fenster wählen wir jetzt unsere Defaults.ffs und bestätigen das ganze. Daraus resultiert folgende Gesamtsituation:



Jetzt das ganze, wie bereits gelernt, speichern und das neu erstellte ROM auf Messages überprüfen. Wenn alles gut gelaufen ist, haben wir jetzt ein Custom angepasstes ROM.

Sind die Dateien



oß, erhalten wir eine Warnung, die wie folgt aussieht: --> das ROM wird nicht gespeichert.

Sowie die Warnung in Messages: `reconstructVolume: root volume can't be grown`. Die Komponenten sind also zu groß und es muss entweder ausgemistet, oder verzichtet werden.

Auf diese Art und Weise könnt ihr sowohl euer ROM personalisieren, als auch Updaten wenn es neue Versionen von bestimmten Komponenten gibt. Erhält zum Beispiel Ozmosis ein Update, oder wird APFS aktualisiert, könnt ihr mit der "Replace as is..."-Funktion ganz einfach euer ROM updaten und daraufhin neu flashen.

~~Soweit erst mal dazu, in nächster Zeit werde ich evtl noch eine Ergänzung veröffentlichen. Bis dahin~~

Weiter gehts: Ozmosis BIOS selber bauen

Beitrag von „Doctor Plagiat“ vom 6. Oktober 2017, 21:15

Obwohl ich kein Oz-taugliches Board habe, schaue ich auch mal über den Tellerrand.

Vielen Dank für den informativen und guten Beitrag, ich freue mich schon auf Teil2.

Beitrag von „Nightflyer“ vom 6. Oktober 2017, 21:37

Interessant, gleich mal abonniert. Vielleicht schaffe ich es ja auch mal, mein eigenes Oz zu backen 

Beitrag von „Noir0SX“ vom 6. Oktober 2017, 21:55

Bis jetzt habe ich mit Oz nix am Hut, naja liegt eher an nichtvorhanden MB, trotzdem vielen Dank für deine investierte Zeit zur Anleitung.

Beitrag von „modzilla“ vom 6. Oktober 2017, 23:04

Respekt! Eine sehr gelungene Erklärung!

Beitrag von „crusher“ vom 6. Oktober 2017, 23:56

Explain how to work apfs for high sierra. How to work apfsjumpstart. Many user say does not see apfs prtition after reset nvram then we must use later shell for boot HS. But users of Ozmosis driver I can say wait to developer fix apfsjumpstart very soon until then use hfs+ format.
Good luck.....

@kuckkuck nice post.

Beitrag von „kuckkuck“ vom 7. Oktober 2017, 00:38

> Zitat von crusher
>
> kuckkuck nice post.

Thanks crusher!

So you think that they'll have the jump start driver working soon? I really wish so, it sucks having a big ass driver in ROM that isn't even supposed to be there... 13 KB would be way better than 271 KB But I haven't read that much about the research going on, there's only some conversation over at applelife that I could find...

Beitrag von „crusher“ vom 7. Oktober 2017, 10:56

I have one patched but he have a diferent error: two line.

Very soon we have fixed apfs. Wait my answer.

Beitrag von „kuckkuck“ vom 12. Oktober 2017, 17:30

Ein ausführlicher Teil 2 des Guides ist jetzt hier zu finden: Post 2

Ich hoffe es hilft dem ein oder anderen 👍

Beitrag von „crusher“ vom 12. Oktober 2017, 21:36

or just follow the video
https://youtu.be/sRAwgXXWXfI

Beitrag von „derHackfan“ vom 12. Oktober 2017, 21:52

> Zitat von kuckkuck
>
> Manche ROMs haben ich von Haus aus genug

Da hast du einen Rechtschreibfehler gemacht.

Beitrag von „griven“ vom 13. Oktober 2017, 23:57

Mir hilft es extrem denn dank dieses Guides und dem von @crusher hinterlegten Videos sollte eigentlich auch jeder selbst in der Lage sein einen OZ Rom zu bauen der ihm taugt
Klassische WIN WIN Situation ich gewinne Freizeit und er User gewinnt an Wissen

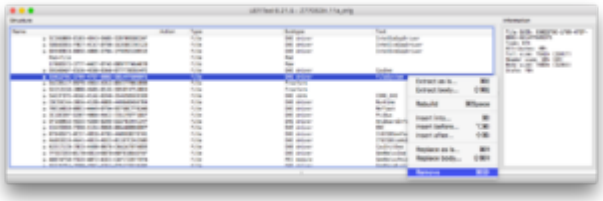
---

Beitrag von „kuckkuck“ vom 14. Oktober 2017, 01:12

Ozmosis BIOS selber bauen

Da mit diesem Guide relativ klar sein sollte, wie man Dateien korrekt zu einem ROM hinzufügt und welche Datei für was zuständig ist, gibt es jetzt noch ein voll ausgestattetes Paket, mit dem sich jeder der will sein eigenes Ozmosis ROM bauen kann.

Eine letzte Sache ist jedoch wichtig! Für die Nutzung von Ozmosis muss aus dem ROM der Filesystem Treiber gelöscht werden! Dieser wird später durch EnhancedFat ersetzt.
Dafür nach `FileSystem` suchen und den DXE Treiber per "Remove" löschen:



Die Schritte um sich sein eigenes ROM mit Ozmosis auszustatten sind danach ganz einfach:

- ROM wie bereits gelernt im UEFI-Tool öffnen
- Nach `CORE_DXE` suchen
- Ans Ende des "CORE_DXE"-Volumen scrollen, letzten Eintrag im Volumen auswählen
- Rechtsklick --> "Insert after..." und jetzt nach und nach alles gewollte hinzufügen
- Speichern und auf Fehler überprüfen

Als erstes sollten dabei unbedingt die bereits benannten Ozmosis Essentials aus dem angehängten Paket eingefügt werden. Erst wenn diese alle untergebracht sind und das ROM gespeichert sowie auf Fehler überprüft wurde, sollten "Luxus-Komponenten" wie die Shell, AcpiPatcher oder Themes hinzugefügt werden. Alle diese Dateien sind im Paket unten zu finden.

Die in diesem Paket enthaltenen .ffs Dateien haben folgende Besonderheiten:

- Ozmosis_HighSierra.ffs: von Cecekpawon gepatched (InsanelyMac), funktioniert mit High Sierra, abwärtskompatibel
- APFS.ffs: APFS von GM Version mit reduziertem TextOutput beim Boot
- defaults.ffs: Allgemeine Defaults.plist mit iMac 14.2 SMBios + neueste FF/FFM, funktioniert mit High Sierra
- FakeSMC.ffs und alle Sensoren: in Version 1759 von Rehabman
- KernextPatcher.ffs: von Cecekpawons Github Seite
- HermitShellX64.ffs: Shell-Version von HermitsCrabsLabs
- EdkShellSmall.ffs: Kleine Version der Shell mit geringerer Auflösung
- OzmosisHorizontalTheme.ffs: Normales HorizontalTheme, Originalversion
- OzmosisHorizontalThemeSmall.ffs: Identisches Theme jedoch ausgemistet und mit besserer Komprimierung, für ROMs mit wenig Platz
- InjectorKext, DisablerKext und PartitionDXE.ffs: Für SnowLeopard
- HFSPlus, EnhancedFat und ExtFS: Ganz normale/altbekannte Version der Dateien

Alle im Paket enthaltenen Komponenten liegen in der Compress-Version vor!

Alle im Paket enthaltenen Komponenten sind abwärts- und High Sierra kompatibel!

Beitrag von „kuckkuck“ vom 15. Oktober 2017, 12:58

Theme tauschen

Gestern habe ich noch einen Thread mit einigen Themes sowie einem Hackintosh-Forum Theme verfasst.
Dazu hier eine kleine Ergänzung:
Wem sein Theme nicht gefällt, oder wer kein Theme hat, kann dieses im ROM tauschen oder auch überhaupt erst hinzufügen, solange genug Platz besteht.

Dafür müsst ihr zuerst einmal ein Theme auswählen, das euch gefällt. Die verschiedenen erhältlichen BootMenü Designs könnt ihr hier finden:
Neue Themes / Ozmosis GUI / BootMenü / UserInterface

Danach besteht die Möglichkeit das ganze entweder, wie in obigem Thread beschrieben, in die EFI zu legen, oder ins ROM zu ergänzen. Da es hier um das UEFI-Tool geht, werde ich von letzterer Möglichkeit reden.

1.
Alle die aus Design- oder anderen Gründen ein anderes UserInterface benutzen wollen, sollten zumindest das unbenutzte Theme aus dem ROM löschen. Dafür wie bereits in diesem Thread beschrieben vorgehen, nach `CORE_DXE` suchen, das `OzmosisHorizontalTheme` finden und bei diesem mit Rechtsklick "**Remove**" auswählen. Danach speichern und fertig ist das altlasten-befreite ROM.

2.
Alle die sich mit ihrer Theme-Wahl sicher sind, können genauso vorgehen, aber anstatt "**Remove**", "**Replace as is...**" auswählen und daraufhin die Theme FFS Datei aus meinem Paket auswählen. Das ganze speichern und schon sind wir fertig.

3.
Alle die gerne ein Theme im ROM hätten, leider aber aus Platzgründen dort keins haben, können versuchen ein kleines Theme hinzuzufügen.
Kleine Themes sind:

- Hackintosh-Forum Theme (normal/16:9) **43 KB**
- Hackintosh-Forum Theme 4:3 **43 KB**
- Horizontal Theme in der neu kompilierten Version von mir **49 KB**

Nur als Vergleich, das normale OzmosisHorizontalTheme ist **207 KB** groß. Nur weil dieses nicht ins ROM passt, heißt es nicht, dass kein Theme ins ROM passt.
Die Themes können ganz einfach hinzugefügt werden: nach `CORE_DXE` suchen, die letzte Datei im gleichen Volumen (oder die Ozmosis-Datei) finden, und bei dieser mit einem Rechtsklick "**Insert after...**" auswählen. Daraufhin die gewollte .ffs aus meinem Paket auswählen, das ROM speichern und hoffen, dass es keine Fehlermeldung aufgrund von Platzproblemen gibt.

Viel Spaß mit den Themes!