

```
1424     "@0,compatible",
1425     Buffer (0x08)
1426     {
1427         "NVDA,NVMac"
1428     },
1429
1430     "@0,device_type",
1431     Buffer (0x08)
1432     {
1433         "display"
1434     },
1435
1436     "@0,name",
1437     Buffer (0x0F)
1438     {
1439         "NVDA,Display-A"
1440     },
1441
1442     "@1,compatible",
1443     Buffer (0x08)
1444     {
1445         "NVDA,NVMac"
1446     },
1447
1448     "@1,device_type",
1449     Buffer (0x08)
1450     {
```

Line: 2293 Column: 41 C++ Tab Size: 4 Defi...

Um Drittanbieter Kexte

zu vermeiden und möglichst viele OS X Funktionen zu nutzen ist das Bearbeiten einer DSDT eine sinnvolle Maßnahme.

DSDT Was ist das?

Ist eine von mehreren im Bios integrierten Tabellen die das System über Hardware spezifische Funktion etc. informiert.

Zum bearbeiten muss diese vom komprimierten ACPI Machine Language (AML) mit Hilfe eines Tools in die ACPI Source Language (ASL) übersetzt werden. Da die meisten Mainboards für Windows ausgelegt werden und entsprechend nur damit getestet werden, kann es bei andern Betriebssystemen, wie Mac OS X (Unix) und Linux, zu Problemen kommen. Eine weitere

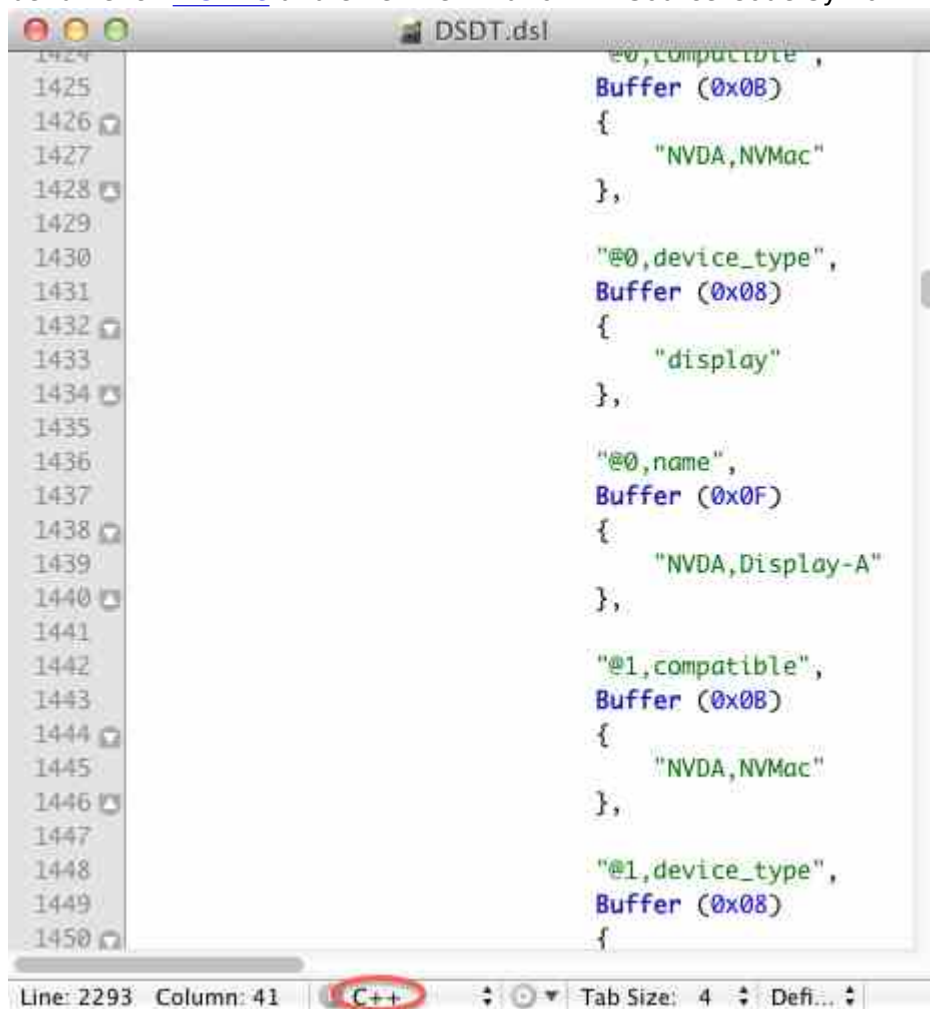
Möglichkeit die für uns Hackintosh Benutzer interessant ist das ändern von Geräte Adressen um dem System kompatible Hardware vorzutauschen (geht natürlich nur bei ähnlicher Hardware).

[Wikipedia Artikel](#)

Tools

Wenn man eine komplett neue DSDT patchen will, z. B nach einem [Bios Update](#) ist der [DSDTFixer](#) ein gutes Tool, er behebt gleich lästige Fehler, um die man sich später nicht mehr kümmern muss.

Mir persönlich gefällt DSDTSE nicht, weil man auf ein sehr kleines Fenster begrenzt ist. Deshalb benutze ich [iASLMe](#) und einen Text Editor mit Source-Code Syntax-Hervorhebung z. B. [TextMate](#)



```
1424      @0,compatible",
1425      Buffer (0x08)
1426      {
1427          "NVDA,NVMac"
1428      },
1429
1430      "@0,device_type",
1431      Buffer (0x08)
1432      {
1433          "display"
1434      },
1435
1436      "@0,name",
1437      Buffer (0x0F)
1438      {
1439          "NVDA,Display-A"
1440      },
1441
1442      "@1,compatible",
1443      Buffer (0x08)
1444      {
1445          "NVDA,NVMac"
1446      },
1447
1448      "@1,device_type",
1449      Buffer (0x08)
1450      {
```

Line: 2293 Column: 41 C++ Tab Size: 4 Defi...

Als Quellcode Art C++ auswählen.

Als Grundlage für Geräte Adressen und anderes sind die [hier](#) erwähnten tools Pflicht.

Tutorials:

1. Grundlagen und compiling Errors

2. [Nvidia-Injection.pdf](#)

3. [BUILT-IN Ethernet.pdf](#)

Da die meisten anderen Sachen wie Sleep (PTS, WAK) schon in den vorgefertigten DSDT's angepasst oder zu spezifisch sind, werde ich darauf zunächst nicht eingehen.

Es kann helfen in DSDT's und IOREG Dumps von original Mac's zu schauen.

Antworten auf diesen Beitrag bleibt zunächst geschlossen, da weitere Informationen Folgen

Beitrag von „iLeopod“ vom 5. Juli 2011, 20:00

DTGP /MCDP

Als Grundlage für die DSM Methoden müssen wir die DTGP Methode deklarieren. Erst mal suchen denn in den vorgefertigten DSDT's ist diese schon vorhanden:

Code

1. Method (DTGP, 5, NotSerialized)
2. {
3. If (LEqual (Arg0, Buffer (0x10)
4. {
5. /* 0000 */ 0xC6, 0xB7, 0xB5, 0xA0, 0x18, 0x13, 0x1C, 0x44,
6. /* 0008 */ 0xB0, 0xC9, 0xFE, 0x69, 0x5E, 0xAF, 0x94, 0x9B
7. }))
8. {

```

9. If (LEqual (Arg1, One))
10. {
11. If (LEqual (Arg2, Zero))
12. {
13. Store (Buffer (One)
14. {
15. 0x03
16. }, Arg4)
17. Return (One)
18. }
19.
20.
21. If (LEqual (Arg2, One))
22. {
23. Return (One)
24. }
25. }
26. }
27.
28.
29. Store (Buffer (One)
30. {
31. 0x00
32. }, Arg4)
33. Return (Zero)
34. }

```

Alles anzeigen

Wenn man eine neue DSDT bearbeitet und wert auf einen schlanken Code legt, kann man auch anstatt der DTGP Methode die MCDP Methode verwenden:

Code

```

1. Method (MCDP, 2, NotSerialized) // New Method V1.1 Ⓓ By Master Chief.
2. {
3. If (LEqual (Arg0, Zero))
4. {
5. Store (Buffer (One)
6. {
7. 0x03
8. }, Arg1)
9. }

```

10. }

Nach jeder DSM Methode hat man dann:

Code

1. MCDP (Arg2, RefOf (Local0))

anstatt:

Code

1. DTGP (Arg0, Arg1, Arg2, Arg3, RefOf (Local0))

Package und Buffer Größen

Code

1. Method (_DSM, 4, NotSerialized)
2. {
3. Store (Package (0x04)
4. {
5. "device-id",
6. Buffer (0x04)
7. {.....}
8. //....

Wir sehen hier beides mal 0x04 wenn wir und dem betreffenden untergeordnetem Bereich etwas ändern stimmt dieser Wert nicht mehr und der Bereich ist wirkungslos. Man kann die Längen mühsam berechnen oder es den compiler machen lassen indem man die Klammern einfach leer lässt:

Code

1. Method (_DSM, 4, NotSerialized)
2. {
3. Store (Package ()
4. {
5. "device-id",
6. Buffer ()
7. {.....}

8. //....

So können auch unnötige Fehlerquellen vermieden werden.

Compiling Errors

.....