

Time machine fix

Creado por EvOSX86 Team

En esta segunda guía, abordaremos como solucionar el problema que algunos usuarios tienen con los backups con Time machine, debido a que el sistema no reconoce la tarjeta de red como dispositivo ENO, para lo cual existe la solución de añadir una cadena EFI al archivo com.apple.boot.plist, pero para los mas "inquietos", os ofreceremos como realizarlo en el DSDT, teniendo asi siempre limpio nuestro com.apple.boot.plist, y de paso, introduciendo la técnica de edición de DSDT para nuestro próximo capítulo (Edición de DSDT para inyectar dispositivos de video).

Herramientas necesarias:

Ioregistry explorer
DSDT patcher.

Contenido:

1. Como reconocer donde está nuestra tarjeta de red en el registro.
2. Descompilación del archivo DSDT.aml.
3. Estructura básica de dispositivos en el DSDT
4. Localización de nuestra tarjeta de red en el archivo DSDT.
5. Inclusión del código.
- 5a. Anadido del código que arregla el problema Time machine.
6. Compilación y prueba.

1. Como reconocer donde está nuestra tarjeta de red en el registro.

Una vez descargado el programa Ioregistry explorer, procedemos ejecutarlo, nos desplazaremos por el arbol de entradas hasta que encontremos la referencia de nuestra tarjeta de red, normalmente suele ser bastante claro:

The screenshot shows the Ioregistry Explorer interface. On the left, a tree view displays the IORegistry hierarchy. The path is: IORegistry -> IOACPIPlatformDevice -> IOACPIPlane -> RP03@1C,2 (Dispositivo madre) -> IOPCI2PCIBridge -> PXS3@0 (Nuestra tarjeta de red). The PXS3@0 device is highlighted in green. On the right, a properties window for the PXS3@0 device is open, showing various properties and their values.

Property	Type	Value
acpi-device	String	IOACPIPlatformDevice is not serializable
acpi-path	String	IOACPIPlane: /_SB/PCI0/RP03@1c0002/PXS3@0
assigned-addresses	Data	<10 00 06 81 00 00 00 00 00 50 00 00 00 00 00 00 00 00 01 00 00 18 00 06 82 00 00 00 00 00 00 00 da 00 00 00 00 00 10 00 00>
built-in	Data	<00>
class-code	Data	<00 00 02 00>
compatible	Data	<"pci104d,9015", "pci10ec,8136", "pciclass,020000">
device-id	Data	<36 81 00 00>
IOChildIndex	Number	0x1
IODeviceMemory	Array	2 values

Como vereis, no está marcada en verde la línea que corresponde a la tarjeta, sino la superior, y tiene su explicación: en el código DSDT no la encontraremos con su nombre, sino con la del dispositivo que la alberga.

Vamos a explicar un poco estos dispositivos:

Marcado en rojo está el dispositivo RP03, este dispositivo alberga un sub-dispositivo denominado PXS3, en el cual se halla nuestra tarjeta de red. A su vez, el dispositivo RP03 es un sub-dispositivo del código correspondiente a los puertos Pci (PCI0)

Cuando añadamos nuestro código, tendremos que estar seguros que es dentro de PXS3 y que cerramos convenientemente el código para no producir errores en la compilación (se verá mas adelante), si os equivocáis y el código cae fuera de PXS3, es muy posible que quede huérfano como dispositivo PCI y que osx no sepa que hacer con el al no estar correctamente definido.

Procedamos ahora a descompilar nuestro DSDT.aml previamente creado.

2. Descompilación del archivo DSDT.aml.

Porqué descompilar el archivo DSDT.aml y no utilizar el DSDT.dsl que tenemos con anterioridad?

Pues porque el DSDT.aml es el archivo que ya contiene todos los parcheos previos (HPET y RTC y corrección de errores), así podemos estar seguros de que funciona (mas que nada porque ya lo habremos probado y estará funcionando).

Paso primero, hacer copia de seguridad del archivo DSDT.aml

Segundo, copia de seguridad de la carpeta debug dentro de la carpeta del parcher (por si acaso)

Mover una copia del archivo DSDT.aml dentro de la carpeta "tools" (que se halla dentro de la carpeta del parcher), estar seguros que se llama DSDT.aml.

Dentro de la carpeta "tools" encontraremos un binario llamado "iasl" bien mi recomendación es que copieis este archivo en /usr/bin con esto lo tendremos siempre disponible al abrir el terminal y no tendremos que estar recordadndo donde lo hemos puesto o arrastrarlo al terminal cada vez que queremos modificar nuestro archivo dst.dsl

Dicho esto, y habiendo copiado el binario en donde hemos mencionado comenzamos con la descompilación del DSDT.aml:

Abrimos terminal, nos logueamos como root y accedemos a la carpeta en donde tenemos almacenado el dsdt (cd "ruta a la carpeta") una vez dentro tecleamos "iasl -d dsdt.aml"

Esto creará el archivo DSDT.dsl, hacemos una copia del mismo para tenerla como respaldo y la movemos fuera de la carpeta tools para no tener superpoblación de archivos.

Ahora ya prodemos editar l DSDT.dsl usando un editor de texto de nuestra elección (se sugiere textmate)

3. Estructura básica de dispositivos en el DSDT

Antes de empezar a insertar código, no está de más que sepamos como es la estructura del código para no perdernos al insertarlo, hechemos un vistazo al código DSDT de nuestra tarjeta de red:

El color rojo pertenece al dispositivo madre, se puede observar el corchete grande verde que abre el código para la definición de los dispositivos, y que no cierra hasta que termina la definición del device PXS3 (la tarjeta de red) y el método de trabajo de la misma.

El color azul oscuro corresponde a la cabecera de nuestra tarjeta de red, su definición está encapsulada por dos corchetes color azul marino.

El color morado corresponde al método que utilizará nuestra tarjeta de red para funcionar (encapsulado en corchetes morados).

Y finalmente un corchete verde que cierra toda la definición del código de RP03.

Debajo encontrareis la definición del primer puerto USB, que es dependiente de PCI0 (que no lo veis porque está mas arriba en el código), y que está fuera de todos nuestros corchetes de apertura y cierre.

Recomiendo que mireis detenidamente el código y lo examineis hasta que veais claro que código está contenido en cada par de corchetes (Relación madre/hijo)

Device (RP03)

```
{
  Name (_ADR, 0x001C0002)
  OperationRegion (P3CS, PCL_Config, 0x40, 0x0100)
  Field (P3CS, AnyAcc, NoLock, WriteAsZeros)
  {
    Offset (0x10),
    LOC3, 1,
    L1C3, 1,
    Offset (0x12),
    , 13,
    LAS3, 1,
    ABR3, 1,
    Offset (0x1A),
    ABR3, 1,
    , 2,
    PDC3, 1,
    , 2,
    PDS3, 1,
    Offset (0x1B),
    LSC3, 1,
    Offset (0x20),
    Offset (0x22),
    PSP3, 1,
    Offset (0x9C),
    , 30,
    HPS3, 1,
    PMS3, 1
  }
}
```

Device (PXS3)

```
{
  Name (_ADR, Zero)
  Name (_PRW, Package (0x02)
  {
    0x09,
    0x03
  })
}
```

Method (_PRT, 0, NotSerialized)

```
{
  # (GPIC)
  {
    Return (Package (0x04)
    {
      Package (0x04)
      {
        0xFFFF,
        Zero,
        Zero,
        0x12
      },
      Package (0x04)
      {
        0xFFFF,
        One,
        Zero,
        0x13
      },
      Package (0x04)
      {
        0xFFFF,
        0x02,
        Zero,
        0x10
      },
    })
  }
}
```

```

Package (0x04)
{
    0xFFFF,
    0x03,
    Zero,
    0x11
}
}
}
}

Device (USB1)
{
    Name (_ADR, 0x001D0000)
    OperationRegion (U1CS, PCI_Config, 0xC4, 0x04)
    Field (U1CS, DWordAcc, NoLock, Preserve)
    {
        U1EN, 2
    }

    Name (_PRW, Package (0x02)
    {
        0x03,
        0x03
    })
    Method (_PSW, 1, NotSerialized)
    {
        If (Arg0)
        {
            Store (0x03, U1EN)
        }
        Else
        {
            Store (Zero, U1EN)
        }
    }

    Method (_S3D, 0, NotSerialized)
    {
        Return (0x02)
    }

    Method (_S4D, 0, NotSerialized)
    {
        Return (0x02)
    }
}

```

4. Localización de nuestra tarjeta de red en el archivo DSDT.

Los ejemplos aquí propuestos están basados en un ordenador portátil Sony, el cual tiene la denominación PXS3 para la tarjeta de red, estad seguros de que en vuestros ordenadores, la denominación será diferente, por lo que tendréis que observar el nombre que tiene en vuestro registro y copiarlo para su búsqueda en el archivo DSDT.dsl

Una vez encontrado nuestro dispositivo de red en el Ioregistry explorer (o el que utilicéis vosotros) abrid el archivo DSDT.dsl con el editor de texto, abrid el menú "buscar" y ponéis el nombre del dispositivo en el campo "búsqueda", en este caso es PXS3, buscadlo hasta que la entrada que aparezca sea del tipo:

[Device \(PXS3\)](#) (o el que os corresponda a vosotros)

Ya tendremos localizado nuestro dispositivo y el dispositivo madre que lo contiene.

5. Inclusión del código.

Una vez que tengamos localizado el sitio correcto, ya solo nos quedará incluir el código que solucionará el problema del Time machine, pero aquí tenemos que parar un momento para explicar un concepto, el código foráneo y el cargador de código foráneo.

Basicamente la explicación es esta:

Nuestro código no es ACPI, es una cadena de información que añadiremos al código ACPI, y si la ponemos tal cual, nuestro registro OSX no la cargará al no ser esta compatible ACPI, para que esta sea cargada, tendremos que utilizar un "cargador", que no es mas que un poco de código que definirá nuestra entrada de código y permitirá su carga por parte del sistema.

Donde insertar el "cargador"?

Esto es sencillo, debéis de buscar con el editor de texto una zona del código que se llama _WAK, y cuando termina la definición de la misma, encajar el código cargador.

Código cargador:

```

Method (DTGP, 5, NotSerialized)
{
    If (LEqual (Arg0, Buffer (0x10)
        {
            /* 0000 */ 0xC6, 0xB7, 0xB5, 0xA0, 0x18, 0x13, 0x1C, 0x44,
            /* 0008 */ 0xB0, 0xC9, 0xFE, 0x69, 0x5E, 0xAF, 0x94, 0x9B
        })))
    {
        If (LEqual (Arg1, One))
        {
            If (LEqual (Arg2, Zero))
            {
                Store (Buffer (One)
                    {
                        0x03
                    }, Arg4)
                Return (One)
            }

            If (LEqual (Arg2, One))
            {
                Return (One)
            }
        }
    }
}

```

```

    }
}
Store (Buffer (One)
{
    0x00
}, Arg4)
Return (Zero)
}

```

Zona de carga:

El código de esta zona dependerá de la máquina, y básicamente tenéis que buscar el final del código `_WAK`, que terminará así:

Return (Package (0x02))

```

{
    Zero,
    Zero
}
}

```

He incluir debajo el código cargador.

EJEMPLO:

Zona de carga mas cargador:

```

Method (_WAK, 1, NotSerialized)
{
    P8XH (One, 0xAB)
    If (LOr (LEqual (Arg0, 0x03), LEqual (Arg0, 0x04)))
    {
        If (And (CFGD, 0x01000000))
        {
            If (LAnd (And (CFGD, 0xF0), LEqual (OSYS, 0x07D1)))
            {
                TRAP (0x3D)
            }
        }
    }

    If (LEqual (RP2D, Zero))
    {
        Notify (\_SB.PCI0.RP02, Zero)
    }

    If (LEqual (Arg0, 0x03)) {}
    If (LEqual (Arg0, 0x04))
    {
        \_SB.PCI0.LPCB.EC0.SELE ()
    }

    P8XH (Zero, 0xCD)
    Return (Package (0x02))
    {
        Zero,
        Zero
    }
}

Method (DTGP, 5, NotSerialized)
{
    If (LEqual (Arg0, Buffer (0x10))
    {
        /* 0000 */ 0xC6, 0xB7, 0xB5, 0xA0, 0x18, 0x13, 0x1C, 0x44,
        /* 0008 */ 0xB0, 0xC9, 0xFE, 0x69, 0x5E, 0xAF, 0x94, 0x9B
    )))
    {
        If (LEqual (Arg1, One))
        {
            If (LEqual (Arg2, Zero))
            {
                Store (Buffer (One)
                {
                    0x03
                }, Arg4)
                Return (One)
            }

            If (LEqual (Arg2, One))
            {
                Return (One)
            }
        }
    }

    Store (Buffer (One)
    {
        0x00
    }, Arg4)
    Return (Zero)
}

```

Salváis el archivo y lo compilamos para ver si se han producido errores en el código.

Abrimos el terminal, nos logueamos como root y accedemos a la carpeta del dsdt (cd "ruta a la carpeta") seguidamente tecleamos

iasl -f dsdt.dsl

Si se producen errores, es que habéis puesto mal el cargador, si ocurren warnings, es que es posible que vengan de antes, no les hagáis mucho caso ahora.

Si no se producen errores, borrard el archivo DSDT.aml que se a generado y proseguimos el trabajo dentro de DSDT.dsl

5a. Añadido del código que arregla el problema Time machine.

Nos desplazamos con la búsqueda dentro del editor de texto a nuestra tarjeta de red (en este caso PXS3)

Device (RP03)

```
{
  Name (_ADR, 0x001C0002)
  OperationRegion (P3CS, PCI_Config, 0x40, 0x0100)
  Field (P3CS, AnyAcc, NoLock, WriteAsZeros)
  {
    Offset (0x10),
    LSC3_1,
    LSC3_2,
    Offset (0x10),
    LSC3_3,
    Offset (0x14),
    ABP3_1,
    PDC3_1,
    PDC3_2,
    PDC3_3,
    Offset (0x1B),
    LSC3_4,
    Offset (0x00),
    Offset (0x02),
    PPS3_1,
    Offset (0x0C),
    SO,
    HPS3_1,
    HPS3_2,
    HPS3_3
  }
}
```

Device (PXS3)

```
{
  Name (_ADR, Zero)
  Name (_PRW, Package (0x02)
  {
    0x09,
    0x03
  })
}
```

Method (_PRT, 0, NotSerialized)

```
{
  If (GPIC)
  {
    Return (Package (0x04)
    {
      Package (0x04)
      {
        0xFFFF,
        Zero,
        Zero,
        0x12
      },
      Package (0x04)
      {
        0xFFFF,
        One,
        Zero,
        0x13
      },
      Package (0x04)
      {
        0xFFFF,
        0x02,
        Zero,
        0x10
      },
      Package (0x04)
      {
        0xFFFF,
        0x03,
        Zero,
        0x11
      }
    })
  }
}
```

Device (USB1)

```
{
  Name (_ADR, 0x001C0000)
  OperationRegion (U1CS, PCI_Config, 0x04, 0x04)
  Field (U1CS, CWordAcc, NoLock, Preserve)
  {
    U1EN, 2
  }
  Name (_PRW, Package (0x02)
  {
    0x03,
    0x03
  })
  Method (_PSW, 1, NotSerialized)
  {
    If (Arg0)
    {
      Store (0x03, U1EN)
    }
    Else
    {
      Store (Zero, U1EN)
    }
  }
  Method (_SDD, 0, NotSerialized)
  {
    Return (0x02)
  }
  Method (_S4D, 0, NotSerialized)
  {
    Return (0x02)
  }
}
```

Ahora que estamos en el sitio correcto, hecharémos un vistazo al código que arregla el problema:

```
Method (_DSM, 4, NotSerialized)
{
```

```

Store (Package (0x06))
{
    "built-in",
    Buffer (One)
    {
        0x01
    },
    "device_type",
    Buffer (0x09)
    {
        "ethernet"
    },
    "name",
    Buffer (0x24)
    {
        "Realtek RTL8111/8168B PCI-E Gigabit"
    }
}, Local0
DTGP (Arg0, Arg1, Arg2, Arg3, RefOf (Local0))
Return (Local0)
}

```

Si os fijáis en las letras en rojo que se sugiere, veréis que están marcadas el Buffer y el nombre de la tarjeta, ¿porque?, pues es simple, deberéis de cambiar el nombre de la tarjeta por el vuestro, y el buffer lo que indica es el número de caracteres (incluyendo los espacios) que hay en el nombre entrecorillado+1, si cambiáis el nombre, pero no el buffer, la compilación fallará.

Como averiguar el buffer (que además está en hexadecimal).
Es bien sencillo, necesitamos un conversor decimal a hexadecimal y al contrario.

Una web que os puede venir muy conveniente es esta:

<http://www.parken.com/apl/HexDecConverter.html>

o esta:

<http://www.paulschou.com/tools/xlate/>

Bien, empecemos:

*nota : Es necesario decir que la línea **"Realtek RTL8111/8168B PCI-E Gigabit"** no es necesaria para fijar TM lo hemos puesto porque nos sirve de ejemplo para que entendamos como se calcula el buffer (explicado a continuación), además nos servirá de información y localización en el ioreg y tb por que no decirlo, queda ciertamente profesional :)

"Realtek RTL8111/8168B PCI-E Gigabit" , el número de letras y espacios contenidos aquí es **35 (decimal)**, por seguridad, añadimos 1, lo que nos da **36**, usando las webs arriba indicadas, ponemos 36 en decimal y nos da como resultado 24 en hexadecimal, osea, el buffer empleado.

Buffer (0x24)

Ahora cambiaremos el nombre que viene en el código que arregla el TM, poniendo en este caso el que le corresponde:

"Realtek RTL8169 PCI-E"

Número de caracteres y espacios +1 = 22 (decimal)

Pasados a **hexadecimal** en las webs arriba indicadas: **16**

Osea, nuestro buffer es:

Buffer (0x16)

Así, el código que arregla el TM para esta tarjeta de red, quedaría:

```

Method (_DSM, 4, NotSerialized)
{
    Store (Package (0x06))
    {
        "built-in",
        Buffer (One)
        {
            0x01
        },
        "device_type",
        Buffer (0x09)
        {
            "ethernet"
        },
        "name",
        Buffer (0x16)
        {
            "Realtek RTL8169 PCI-E"
        }
    }, Local0
    DTGP (Arg0, Arg1, Arg2, Arg3, RefOf (Local0))
    Return (Local0)
}

```

Lo cual ya es correcto y está listo para su inserción en el código DSDT.

A propósito.... fijaros en las últimas líneas de código:

DTGP (Arg0, Arg1, Arg2, Arg3, RefOf (Local0))
Return (Local0)

Estas líneas son las que permiten que el código cargador identifique que es "nuestro" código y las cargue, si no estuviesen estas líneas, el código cargador no sabría que las tiene que cargar.

Insertemos nuestro código ahora que está correcto:

Device (RP03)

```
{
  Name (_ADR, 0x001C0002)
  OperationRegion (P3CS, PCL_Config, 0x40, 0x0100)
  Field (P3CS, AnyAcc, NoLock, WriteAsZeros)
  {
    Offset (0x10),
    LSCL, 1,
    Offset (0x12),
    LS,
    Offset (0x14),
    ABP0, 1,
    PCGA, 1,
    PCGB, 1,
    Offset (0x1B),
    LSCL, 1,
    Offset (0x20),
    P3P0, 1,
    Offset (0x22),
    P3P1, 1,
    Offset (0x2C),
    SO,
    HPISA, 1,
    PMS0, 1
  }
}
```

Device (PXS3)

```
{
  Name (_ADR, Zero)
  Name (_PRW, Package (0x02)
  {
    0x09,
    0x03
  })
  Method (_DSM, 4, NotSerialized)
  {
    Store (Package (0x06)
    {
      "built-in",
      Buffer (One)
      {
        0x01
      },
      "device_type",
      Buffer (0x09)
      {
        "ethernet"
      },
      "name",
      Buffer (0x16)
      {
        "Realtek RTL8169 PCI-E"
      }
    }, Local0)
    DTGP (Arg0, Arg1, Arg2, Arg3, RefOf (Local0))
    Return (Local0)
  }
}
```

Method (_PRT, 0, NotSerialized)

```
{
  If (GPIC)
  {
    Return (Package (0x04)
    {
      Package (0x04)
      {
        0xFFFF,
        Zero,
        Zero,
        0x12
      },
      Package (0x04)
      {
        0xFFFF,
        One,
        Zero,
        0x13
      },
      Package (0x04)
      {
        0xFFFF,
        0x02,
        Zero,
        0x10
      },
      Package (0x04)
      {
        0xFFFF,
        0x03,
        Zero,
        0x11
      }
    })
  }
}
```

```
}  
}
```

Como podréis observar, el código está insertado dentro de los corchetes **azul celeste**, que definen la tarjeta de red, especificando así para OSX los parámetros requeridos para que Time machine funcione correctamente y debajo de:

```
Name (_ADR, Zero)  
    Name (_PRW, Package (0x02)  
    {  
        0x09,  
        0x03  
    })
```

Que definen nuestra tarjeta de red.

6. Compilación y prueba.

Ahora solo nos quedará salvar el archivo y compilarlo para ver si tiene errores de sintaxis o no. (visto previamente)

Si los tiene, aseguraos de que no hayan quedado corchetes huérfanos o borrados.

Si no los tiene, renombrad el archivo DSDT.aml que tenéis en el root del sistema a "DSDTold.aml" y copiad el nuevo en la raíz del disco.

Si fallase el nuevo DSDT.aml, (por ejemplo se congela el sistema), podéis arrancar el sistema utilizando el comando DSDT=DSDTold.aml (introducido cuando veáis el arranque chameleon presionando F8 y seleccionando la instalación de Leopard).

Esperamos que os haya sido útil este segundo capítulo de la edición de DSDT.

Guía by: Roisoft y Pere