

Formateo de la vista y personalización

En el capítulo Descubra PowerShell hemos visto que el resultado de un commandlet devuelve casi siempre un conjunto de propiedades (que llamaremos «propiedades por defecto») formateado generalmente en forma de tabla o de lista.

Un de los puntos fuertes de PowerShell es que nos ofrece la posibilidad de modificar el juego de valores mostrados por defecto, y no solamente para cada comando, sino también para cada tipo. En efecto, se podría pensar que los valores por defecto que aparecen durante la ejecución de un comando son propios de los comandos; pero ese no es en absoluto el caso. Es el tipo de objeto (a mostrar) quien determinará su formateo.

En realidad, cuando ejecute una commandlet en la consola PowerShell, el flujo de objetos resultantes del comando se transmite al commandlet `out-Default` vía la tubería. Esto es así para todos los comandos (que muestren algo por pantalla) que pueda introducir en el interprete de comandos.


`out-Default` es responsable de la visualización y el formateo de los flujos de objetos. Si el flujo de objetos es de tipo cadena, entonces `out-Default` lo pasa directamente, siempre por la tubería, a el commandlet `out-Host`. A la inversa, si el flujo no contiene cadenas, entonces `out-Default` inspecciona el objeto y determina qué debe hacer con él.

En primer lugar, Powershell determinará el tipo del objeto y tratará de encontrarle una vista predefinida. Las vistas predefinidas se describen en un archivo XML, cuyo nombre es del tipo `*.format.ps1xml`. Veremos en las próximas páginas cómo modificarlas o crear otras nuevas. Por tanto, si una vista existe para el tipo de objeto en cuestión, entonces éste estará formateado en función de la definición de la vista. En otras palabras, si la definición de la vista es una tabla, entonces `out-Default` transmitirá el flujo del objeto al commandlet de formateo adecuado (tal como `Format-Table`, `Format-List` o `Format-Wide`), es decir, en este caso, `Format-Table`.

Observe que la ejecución de todos estos comandos nos da exactamente el mismo resultado:

```
Get-ChildItem
Get-ChildItem | Out-Default
Get-ChildItem | Format-Table
Get-ChildItem | Format-Table | Out-Host
Get-ChildItem | Format-Table | Out-String | Out-Host
Get-ChildItem | Format-Table | Out-String | Out-Default
```

Pero si no hay una vista predefinida para el objeto que queremos mostrar, `out-Default` busca el primer objeto en el flujo y cuenta el número de propiedades que tiene. Si el objeto posee cinco o más, `out-Default` envía entonces el flujo a `Format-List`, sino lo envía a `Format-Table`. Cuando el flujo de objetos se trasmite a `Format-Table`, este comando tendrá que generar las columnas. Para ello, creará tantas columnas (menos de cinco por tanto) como propiedades posea el primer objetivo del flujo. Por ejemplo, si el primer objetivo del flujo posee tres propiedades, entonces la tabla tendrá tres columnas, aunque el segundo objeto posea diez propiedades. En este caso, tendremos un problema para mostrar los otros objetos.

 Para saber más acerca del formateo de los objetos, puede consultar el enlace siguiente: <http://blogs.msdn.com/powershell/archive/2006/04/30/586973.aspx>. Se trata de una explicación detallada por parte de **Jeffrey Snover**, el arquitecto de PowerShell.

1. Descubrimiento de los ficheros de formateo por defecto

Como usted ha podido comprender, PowerShell se ha entregado con una vista predefinida para cada tipo de objeto. Los archivos de definición de formateo se encuentran en el mismo directorio que el archivo de definición de los tipos. El directorio de instalación de PowerShell (`%systemroot%\system32\windowspowershell\v1.0` o también llamado `$PSHOME`).

Estos archivos son los siguientes (los archivos con un asterisco son específicos de la versión 2 de PowerShell):

- **Certificate.format.ps1xml**
- **Diagnostics.Format.ps1xml (*)**
- **DotNetTypes.format.ps1xml**
- **FileSystem.format.ps1xml**
- **Getevent.type.ps1xml (*)**
- **Help.format.ps1xml**
- **PowerShellCore.format.ps1xml**
- **PowerShellTrace.format.ps1xml**
- **Registry.format.ps1xml**
- **WSManFormat.ps1xml (*)**

Los archivos XML poseen una nomenclatura (o esquema XML) propia de PowerShell. Le invitamos a ver uno de cerca para familiarizarse un poco con su sintaxis tan especial.

He aquí la estructura general de este archivo:

Dpto. de INFORMATICA - dc800154-6e6b-44-8610-0cda54a5c097

```

<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
  <ViewDefinitions>
    <View>
      <Name> Nombre de la vista </Name>
      <ViewSelectedBy> Nombre del tipo de objeto a definir </Type>
      <TypeName>
      </ViewSelectedBy>
      <GroupBy>
        <PropertyName> Nombre de una propiedad de reagrupamiento </PropertyName>
        <Label> Nombre de la propiedad </Label>
      </GroupBy>
    <TableControl>
      <TableHeaders>
        <TableColumnHeader>
          <Width> Tamaño de la columna </Width>
          <Label> Título de la columna </Label>
          <Alignment> Posición </Alignment>
        </TableColumnHeader>
        <TableColumnHeader/>
      </TableHeaders>
      <TableRowEntries>
        <TableRowEntry>
          <TableColumnItems>
            <TableColumnItem>
              <PropertyName> Nombre de la propiedad </PropertyName>
            </TableColumnItem>
            <TableColumnItem>
              <PropertyName> Nombre de la propiedad </PropertyName>
            </TableColumnItem>
          </TableColumnItems>
        </TableRowEntry>
      </TableRowEntries>
    </TableControl>
  </View>
</ViewDefinitions>
</Configuration>

```

Definición del nombre de la vista

Definición de las cabeceras de la columna

Definición del contenido de las columnas

Algunas explicaciones sobre la estructura:

La primera línea contiene la cabecera estándar de un archivo XML; en ella se define su versión y su codificación. A continuación vienen los elementos raíz `Configuration` y `viewDefinitions`. Después por cada nueva vista a definir aparece un elemento `view`. El elemento `name` contiene el nombre de la vista. `viewSelectedBy` contiene el nombre del o de los tipos a definir. El elemento `typeName` especifica todos los tipos. El nodo `groupBy` indica la propiedad de reagrupamiento de los objetos.

Acto seguido tenemos la definición de la tabla. La cabecera de columnas se define en primer lugar con el elemento `TableColumnHeader` en el que se especifica: el tamaño de la columna (en número de caracteres), su título, así como la alineación de los datos a mostrar (left o right). Se definen tanto los encabezados de las columnas como las propiedades a mostrar. Si no hay nada indicado en el interior de un elemento `TableColumnHeader` (como en la imagen anterior `<TableColumnHeader/>` equivale a `<TableColumnHeader> </TableColumnHeader>`), entonces el título de la columna toma el

nombre de la propiedad y el tamaño se ajusta al contenido.

➤ Se recomienda poner nombre a las columnas (el título) exactamente como los nombres de las propiedades si corresponden a una propiedad, para que sea más claro para el usuario. Por ejemplo, si una propiedad se llama `ProcessName`, llamar a la columna `ProcessName` y no `Name`, o `Process`, o `Process Name` con un espacio. El cambio de nombre de las propiedades con nombres más fáciles de usar puede dejarse a elección del usuario en el tratamiento final, con las opciones avanzadas de los comandos de formateo.

Y para finalizar, el contenido de las columnas está definido en el elemento `TableColumnItem`. Para ello, pueden utilizarse los elementos `PropertyName` o `ScriptBlock`.

`PropertyName` sirve para indicar el nombre de la propiedad a mostrar. Mientras que `scriptBlock` permite hacer mucho más, como por ejemplo aplicar un tratamiento a una propiedad. Por este vía, es posible (entre otros) formatear una fecha, convertir un tamaño de ficheros a Kilobytes o incluso mostrar una propiedad en color, etc.

2. Creación de un archivo de formateo personalizado

Para comprender mejor el funcionamiento de los archivos de formateo, vamos a tomar un caso concreto: la vista del commandlet `Get-ChildItem`. A pesar de que nos proporciona diariamente un valioso servicio, podría mejorar mucho.

`Get-ChildItem` nos reenvía por defecto un cierto número de propiedades: `Mode`, `LastWriteTime`, `Length` y `Name`.

```
PS > Get-ChildItem $PSHOME

    Directorio: C:\Windows\System32
\WindowsPowerShell\v1.0

Mode                LastWriteTime         Length Name
----                -
d-----          14/07/2009         06:56      en-US
d-----          14/07/2009         06:52    Examples
d-----          04/09/2009         11:19      es-ES
d-----          14/07/2009         09:49    Modules
-a----          10/06/2009         23:24    27338 Certificate.format.ps1xml
-a----          14/07/2009         03:06   126976 CompiledComposition.Microsoft...
-a----          10/06/2009         23:24    27106 Diagnostics.Format.ps1xml
-a----          10/06/2009         23:24    72654 DotNetTypes.format.ps1xml
-a----          10/06/2009         23:24    24857 FileSystem.format.ps1xml
-a----          10/06/2009         23:24    15603 getevent.types.ps1xml
-a----          10/06/2009         23:24   257847 Help.format.ps1xml
-a----          14/07/2009         03:14   452608 powershell.exe
-a----          10/06/2009         23:24    89703 PowerShellCore.format.ps1xml
-a----          10/06/2009         23:24    18612 PowerShellTrace.format.ps1xml
-a----          14/07/2009         03:23   204800 powershell_ise.exe
-a----          14/07/2009         03:06    20480 PSEvents.dll
-a----          14/07/2009         03:23   154624 pspluginwkr.dll
-a----          14/07/2009         03:06     2048 pwrshmsg.dll
-a----          14/07/2009         03:15    24064 pwrshsip.dll
-a----          10/06/2009         23:24    20120 Registry.format.ps1xml
-a----          10/06/2009         23:24   168372 types - Copia.ps1xml
-a----          10/06/2009         23:24   168372 types.ps1xml
-a----          10/06/2009         23:24    24498 WSMAN.Format.ps1xml
```

Sería especialmente agradable tener el tamaño de los archivos en Kilobytes (Kb), ya que se leerá con dificultad tan

pronto como la cifra pase a ser muy grande, así como la fecha de creación de los archivos y directorios. Y ya estamos, ¿por qué no tratar de traducir el título de las columnas al español (atención, esta no es una buena práctica - véase la nota anterior - pero nos permite mostrar todo lo que se puede hacer)?

Para no partir de cero, empecemos la búsqueda del archivo de formato que define los objetos de tipo `FileSystem`. Afortunadamente, en el directorio de instalación de PowerShell, existe un archivo que se llama `FileSystem.format.ps1xml`.

Para facilitarnos la tarea, podríamos tener ganas de modificar este fichero directamente, pero sería una muy mala idea. Por una parte, podría perjudicar el buen funcionamiento general de PowerShell, y por otra parte, podríamos tener problemas con la seguridad (ino con la policía, puede estar tranquilo! ☺). En efecto, este archivo ha sido firmado digitalmente y si realizamos cualquier modificación, la firma dejará de corresponder con archivo original y PowerShell podría negarse a funcionar. Por tanto, trabajaremos sobre una copia del archivo original. Por supuesto, debemos eliminar la firma digital. Para el resto, nos contentaremos con modificarlo.

Añadamos a la definición de la cabecera de la tabla un elemento `<TableColumnHeader>` (que corresponderá a la columna `CreationTime`) entre el que define la propiedad `Mode` y la propiedad `LastWriteTime`.

Antes:

```
<TableHeaders>
  <TableColumnHeader>
    <Label>Mode</Label>
    <Width>7</Width>
    <Alignment>left</Alignment>
  </TableColumnHeader>
  <TableColumnHeader>
    <Label>LastWriteTime</Label>
    <Width>25</Width>
    <Alignment>right</Alignment>
  </TableColumnHeader>
  <TableColumnHeader>
    <Label>Length</Label>
    <Width>10</Width>
    <Alignment>right</Alignment>
  </TableColumnHeader>
  <TableColumnHeader/>
</TableHeaders>
```

Después:

```
<TableHeaders>
  <TableColumnHeader>
    <Label>Mode</Label>
    <Width>7</Width>
    <Alignment>left</Alignment>
  </TableColumnHeader>
  <TableColumnHeader>
    <Label>CreationTime</Label>
    <Width>25</Width>
    <Alignment>right</Alignment>
  </TableColumnHeader>
  <TableColumnHeader>
    <Label>LastWriteTime</Label>
    <Width>25</Width>
    <Alignment>right</Alignment>
  </TableColumnHeader>
```

```

        <TableColumnHeader>
            <Label>Length</Label>
            <Width>10</Width>
            <Alignment>right</Alignment>
        </TableColumnHeader>
    </TableColumnHeader/>
</TableHeaders>

```

Ahora debemos modificar la definición del contenido de las columnas, añadiendo - siempre justo después de `mode` - el contenido de la propiedad que acabamos de crear. Sustituyamos igualmente la propiedad `length` por un bloque de script. Éste nos va a hacer la conversión bytes -> Kilobytes y nos añadirá «Kb» en el valor de la propiedad. Como a continuación:

Antes:

```

<TableRowEntries>
    <TableRowEntry>
        <Wrap/>
        <TableColumnItems>
            <TableColumnItem>
                <PropertyName>Mode</PropertyName>
            </TableColumnItem>
            <TableColumnItem>
                <ScriptBlock>
                    [String]::Format("{0,10} {1,8}",
                        $_.LastWriteTime.ToString("d"),
                        $_.LastWriteTime.ToString("t"))
                </ScriptBlock>
            </TableColumnItem>
            <TableColumnItem>
                <PropertyName>Length</PropertyName>
            </TableColumnItem>
            <TableColumnItem>
                <PropertyName>Name</PropertyName>
            </TableColumnItem>
        </TableColumnItems>
    </TableRowEntry>
</TableRowEntries>

```

Después:

```

<TableRowEntries>
    <TableRowEntry>
        <Wrap/>
        <TableColumnItems>
            <TableColumnItem>
                <PropertyName>Mode</PropertyName>
            </TableColumnItem>
            <TableColumnItem>
                <PropertyName>CreationTime</PropertyName>
            </TableColumnItem>
            <TableColumnItem>
                <ScriptBlock>
                    [String]::Format("{0,10} {1,8}",
                        $_.LastWriteTime.ToString("d"),
                        $_.LastWriteTime.ToString("t"))
                </ScriptBlock>
            </TableColumnItem>
        </TableColumnItems>
    </TableRowEntry>
</TableRowEntries>

```

```

        </ScriptBlock>
    </TableColumnItem>
    <TableColumnItem>
        <ScriptBlock>
            $a = [math]::round($_.length/1024,0)
            if ($a -gt 0) {
                [string]$a += " Kb"
            }
            else {
                $a = ""
            }
            $a
        </ScriptBlock>
    </TableColumnItem>
</TableColumnItem>
    <PropertyName>Name</PropertyName>
</TableColumnItem>
</TableColumnItems>
</TableRowEntry>
</TableRowEntries>

```

En estos momentos, no nos queda más que dar a conocer este nuevo archivo de formateo a PowerShell. Suponiendo que usted haya llamado a su archivo **perso.format.ps1xml**, utilice el comando siguiente:

```
PS > Update-FormatData -Prepend perso.format.ps1xml
```

el parámetro **-Prepend** indica a PowerShell que debe utilizar prioritariamente este archivo en lugar del nativo.


Vamos, veremos si nuestras modificaciones han cambiado algo:

```
PS > Get-Childitem $PSHOME
```

```
Directorio : C:\Windows\System32\WindowsPowerShell\v1.0
```

Mode	CreationTime	LastWriteTime	Length	Name
d----	14/07/2009 06:52:30	14/07/2009 06:52		Examples
d----	14/07/2009 10:39:37	14/07/2009 10:39		es-ES
d----	14/07/2009 06:52:30	14/07/2009 11:01		Modules
-a---	13/07/2009 22:34:42	10/06/2009 23:24	27 Kb	Certificate.format...
-a---	13/07/2009 22:34:42	10/06/2009 23:24	26 Kb	Diagnostics.Format...
-a---	13/07/2009 22:34:42	10/06/2009 23:24	71 Kb	DotNetTypes.format...
-a---	13/07/2009 22:34:42	10/06/2009 23:24	24 Kb	FileSystem.format...
-a---	13/07/2009 22:34:42	10/06/2009 23:24	15 Kb	getevent.types.ps1xml
-a---	13/07/2009 22:34:42	10/06/2009 23:24	252 Kb	Help.format.ps1xml
-a---	14/07/2009 01:32:37	14/07/2009 03:14	442 Kb	powershell.exe
-a---	13/07/2009 23:47:02	14/07/2009 03:23	200 Kb	powershell_ise.exe
-a---	14/07/2009 01:32:28	14/07/2009 03:06	20 Kb	PSEvents.dll
-a---	14/07/2009 01:32:33	14/07/2009 03:23	151 Kb	pspluginwkr.dll
-a---	14/07/2009 01:32:29	14/07/2009 03:06	2 Kb	pwrshmsg.dll
-a---	14/07/2009 01:32:28	14/07/2009 03:15	24 Kb	pwrshsip.dll
-a---	13/07/2009 22:34:42	10/06/2009 23:24	20 Kb	Registry.format.ps1xml
-a---	10/06/2009 23:24:31	10/06/2009 23:24	164 Kb	types.ps1xml
-a---	13/07/2009 22:34:42	10/06/2009 23:24	24 Kb	WSMan.Format.ps1xml

¿No es simplemente fabuloso?

 Aunque factible, no se recomienda modificar la propiedad `Length` como hemos hecho nosotros. En efecto, añadiendo la unidad «Kb» al valor, hemos modificado su tipo. Anteriormente la propiedad `Length` era de tipo `int`, y ahora es de tipo `String`. Por consiguiente ya no podemos realizar fácilmente las pruebas sobre el tamaño de los archivos.

Como habíamos convenido, podemos cambiar el título de las columnas modificando el elemento `<Label>` contenido en el elemento `<TableHeaders>`, como a se muestra a continuación:

```
<TableHeaders>
  <TableColumnHeader>
    <Label>Mode</Label>
    <Width>7</Width>
    <Alignment>left</Alignment>
  </TableColumnHeader>
  <TableColumnHeader>
    <Label>Fecha de creación</Label>
    <Width>25</Width>
    <Alignment>right</Alignment>
  </TableColumnHeader>
  <TableColumnHeader>
    <Label>Fecha de modificación</Label>
    <Width>25</Width>
    <Alignment>right</Alignment>
  </TableColumnHeader>
  <TableColumnHeader>
    <Label>Tamaño</Label>
    <Width>10</Width>
    <Alignment>right</Alignment>
  </TableColumnHeader>
  <TableColumnHeader/>
</TableHeaders>
```


Resultado:

```
PS > Get-Childitem $PSHOME
```

```
Directorio : C:\Windows\System32\WindowsPowerShell\v1.0
```


Mode	Fecha de creación	Fecha de modificación	Tamaño	Name
d----	14/07/2009 06:52:30	14/07/2009 06:52		Examples
d----	14/07/2009 10:39:37	14/07/2009 10:39		es-ES
d----	14/07/2009 06:52:30	14/07/2009 11:01		Modules
-a---	13/07/2009 22:34:42	10/06/2009 23:24	27 Kb	Certificate.form...
-a---	13/07/2009 22:34:42	10/06/2009 23:24	26 Kb	Diagnostics.Form...
-a---	13/07/2009 22:34:42	10/06/2009 23:24	71 Kb	DotNetTypes.form...
-a---	13/07/2009 22:34:42	10/06/2009 23:24	24 Kb	FileSystem.forma...
-a---	13/07/2009 22:34:42	10/06/2009 23:24	15 Kb	getevent.types.p...
-a---	13/07/2009 22:34:42	10/06/2009 23:24	252 Kb	Help.format.ps1xml
-a---	14/07/2009 01:32:37	14/07/2009 03:14	442 Kb	powershell.exe
-a---	13/07/2009 23:47:02	14/07/2009 03:23	200 Kb	powershell_ise.exe
-a---	14/07/2009 01:32:28	14/07/2009 03:06	20 Kb	PSEvents.dll
-a---	14/07/2009 01:32:33	14/07/2009 03:23	151 Kb	pspluginwkr.dll
-a---	14/07/2009 01:32:29	14/07/2009 03:06	2 Kb	pwrshmsg.dll


```
-a--- 14/07/2009 01:32:28 14/07/2009 03:15 24 Kb pwrshsip.dll
-a--- 13/07/2009 22:34:42 10/06/2009 23:24 20 Kb Registry.format...
-a--- 10/06/2009 23:24:31 10/06/2009 23:24 164 Kb types.ps1xml
-a--- 13/07/2009 22:34:42 10/06/2009 23:24 24 Kb WSMAN.Format.ps1xml
```

 Para que esto funcione y los acentos de nuestras propiedades se muestren correctamente, será necesario modificar el tipo de codificación en la primera línea del archivo ps1xml, precisando **UTF-16** en lugar de **UTF-8**. **<?xml version="1.0" encoding="utf-16" ?>**. No olvide guardar su archivo en Unicode **UTF-16**. En la siguiente sección de este capítulo, se dan más detalles sobre el formato Unicode.

Por último, siguiendo con los archivos de formateo podremos mostrar fácilmente el nombre de los archivos de un color, y los directorios de otro distinto, o mejor aún, asignar un color en función de la extensión del archivo. En resumen, ¡no existen límites!

Hemos basado todos nuestros ejemplo en el tipo `FileSystem` pero debe saber que puede crear vistas personalizadas para cualquier otro tipo.

 Para conocer más acerca del formateo de los objetos, puede consultar el enlace siguiente («extending object types and formatting»): <http://msdn.microsoft.com/es-es/library/ms714665.aspx>