

Adición de métodos y propiedades personalizadas

Como hemos comentado en la introducción, PowerShell es extensible. Vamos a ver ahora cómo añadir nuevas propiedades y métodos a los tipos de datos. Ya que ¿Hay algo más frustrante que listar un gran número de propiedades y no encontrar aquella que buscamos? Ahora ya no le volverá a suceder, ¡gracias a PowerShell podrá añadirlas usted mismo!

Un ejemplo para ilustrar nuestras palabras. Cuando utilice el commandlet `Get-Member` en un archivo o en una carpeta, obtendrá una lista de propiedades y métodos asociados. Exactamente 77 (69 con PowerShell v1) para un archivo, y 65 (58 con PowerShell v1) para una carpeta (gracias a los comandos `Get-Item miArchivo | Get-Member -force | Measure-Object`).

Naturalmente, la propiedad que buscábamos no está (¡siempre igual! 😊): hubiésemos querido conocer el propietario de un archivo.

¡Que no cunda el pánico! Empecemos por listar los métodos y propiedades de un archivo tecleando el comando siguiente:

```
PS > Get-Item miArchivo.txt | Get-Member -Force
```

Name	MemberType	Definition

Mode	CodeProperty	System.String Mode{get=Mode;}
pstypenames	CodeProperty	System.Collections.ObjectModel...
psadapted	MemberSet	psadapted {Name, Length, Direct..
PSBase	MemberSet	PSBase {Name, Length, Directory...
psextended	MemberSet	psextended {PSPath, PSParentPat...
psobject	MemberSet	psobject {Members, Properties,...
PSStandardMembers	MemberSet	PSStandardMembers {DefaultDispl...
AppendText	Method	System.IO.StreamWriter AppendTe...
CopyTo	Method	System.IO.FileInfo CopyTo(strin...
Create	Method	System.IO.FileStream Create()
CreateObjRef	Method	System.Runtime.Remoting.ObjRef...
CreateText	Method	System.IO.StreamWriter CreateTe...
Decrypt	Method	System.Void Decrypt()
Delete	Method	System.Void Delete()
Encrypt	Method	System.Void Encrypt()
Equals	Method	bool Equals(System.Object obj)
GetAccessControl	Method	System.Security.AccessControl.F...
GetHashCode	Method	int GetHashCode()
GetLifetimeService	Method	System.Object GetLifetimeServic...
GetObjectData	Method	System.Void GetObjectData(Syste...
GetType	Method	type GetType()
get_Attributes	Method	System.IO.FileAttributes get_At...
get_CreationTime	Method	System.DateTime get_CreationTim...
get_CreationTimeUtc	Method	System.DateTime get_CreationTim...
get_Directory	Method	System.IO.DirectoryInfo get_Dir...

Si se examina de cerca, podemos observar el método `GetAccessControl`. Éste posee un nombre muy interesante, y elaborando un poco este método, acabará por darnos la información que buscamos...

Lista de las propiedades y métodos que empiecen por «get» asociadas a la clase `getAccessControl`:

```
PS > (Get-Item miArchivo.txt).getAccessControl() | Get-Member |  
where {$_.name -like "get*"}  
  
TypeName: System.Security.AccessControl.FileSecurity
```

Name	MemberType	Definition
----	-----	-----
GetAccessRules	Method	System.Security.AccessContro...
GetAuditRules	Method	System.Security.AccessContro...
GetGroup	Method	System.Security.Principal.Id...
GetHashCode	Method	System.Int32 GetHashCode()
GetOwner	Method	System.Security.Principal.Id...
.....		

Observamos que hay un método (**GetOwner**) que posee un nombre similar al que buscamos.

Entonces probaremos lo siguiente:

```
PS > (Get-Item miArchivo.txt).getAccessControl().GetOwner()

Sobrecarga no encontrada para «GetOwner» y número de argumentos «0».
A nivel de línea: 1 Carácter: 54
+ (Get-Item miArchivo.txt).getAccessControl().GetOwner( << )
```

Hubiera sido demasiado sencillo, ¡y esta línea de comandos nos devuelve un mensaje de error muy simpático! En efecto, si se mira más detalladamente la definición de este método:


```
PS > (Get-Item miArchivo.txt).getAccessControl() | Get-Member |
where {$_.name -eq "getOwner"} | format-list

TypeName      : System.Security.AccessControl.FileSecurity
Name           : GetOwner
MemberType    : Method
Definition    : System.Security.Principal.IdentityReference GetOwner(Type
targetType)
```

Se puede ver que está esperando que se le pase un parámetro de tipo `targetType`.

Vamos a necesitar un poco de ayuda para encontrar los tipos esperados, ya que éstos no se encuentran en la ayuda estándar de PowerShell. Es normal, ya que estamos manipulando directamente objetos del Framework .NET.

En esta fase, sólo nos queda una cosa por hacer: ir a consultar la ayuda directamente en la web de Microsoft y más concretamente, en la base de conocimientos MSDN.

 Para obtener la ayuda sobre las clases de objetos de Framework .NET, utilice la URL siguiente: <http://msdn.microsoft.com/es-es/default.aspx> y escriba el nombre de la clase que buscamos en el campo **Buscar MSDN**, en la parte superior central de la página.

Después de haber buscado la información en el sitio web MSDN hemos descubierto que la clase `IdentityReference` espera por parámetro las clases `NTAccount` o `SecurityIdentifier`.

■ Probemos entonces lo siguiente:

```
PS > (Get-Item miArchivo.txt).getAccessControl().GetOwner(`
[System.Security.Principal.NTAccount])

Value
-----
wrkoscar\Oscar
```

¡Funciona! Recuperamos el nombre del propietario del archivo en cuestión, así como el nombre de dominio relacionado con su cuenta (en nuestro caso wrkoscar).

➤ Cuando hagamos la llamada a un tipo o a una clase de objeto .NET, no olvide especificarla entre corchetes, como en el ejemplo siguiente: `[System.Security.Principal.NTAccount]`

Veamos ahora lo que nos devuelve el comando si se le especifica la clase `SecurityIdentifier`:

```
PS > (Get-Item miArchivo.txt).getAccessControl().GetOwner(`
[System.Security.Principal.SecurityIdentifier]) | Format-List

BinaryLength      : 28
AccountDomainSid   : S-1-5-21-2069618812-4153402021-1334178849
Value              : S-1-5-21-2069618812-4153402021-1334178849-1002
```

Esta vez hemos recuperado dos SID (*Security Identifier*): el SID correspondiente al dominio al que el usuario pertenece, y el SID del usuario.

Bien, centrémonos en el tema de esta parte que, no lo olvidemos, hace referencia a la extensión del conjunto de propiedades y métodos de un tipo determinado. Sabemos ahora cómo obtener la información «propietario de un archivo», pero es tan larga y complicada de escribir que corremos el riesgo de no utilizarlo cada día. Vamos entonces a crear una propiedad adicional para el tipo de archivo (`System.IO.FileInfo`).

Se realiza en varias etapas:

- Creación de un archivo XML que describa las nuevas propiedades (y métodos si los hay).
- Importación de este archivo en PowerShell (utilizando el comando `Update-TypeData`).

1. Creación del archivo de definición del tipo

Antes de empezar, debe saber que en PowerShell todos los tipos existentes están definidos en el archivo `types.ps1xml`. Puede encontrar este archivo en el directorio `%windir%\system32\windowspowershell\v1.0` para los entornos de 32 bits y en `%windir%\syswow64\WindowsPowerShell\v1.0` para los sistemas de 64 bits (los más atentos observarán de paso que esta ruta no es ni más ni menos que el contenido de la variable `$PSHOME`). Se trata de un archivo XML que se puede abrir con el bloc de notas. Para ver el contenido, le aconsejamos hacer una copia y cambiar la extensión a `.xml`. Así se abrirá automáticamente con Internet Explorer, y podrá obtener la coloración sintáctica y mucho más... Este archivo XML posee una nomenclatura (o esquema XML) propia.

➤ Aunque sea posible modificar directamente el fichero `types.ps1xml`, es muy desaconsejable hacerlo ya que puede provocar un funcionamiento erróneo de PowerShell.

A fin de añadir nuestra propiedad `owner`, tenemos que crear un nuevo archivo `ps1xml`. Deberá crearlo en el mismo lugar que el archivo del tipo por defecto. Llamémosle por ejemplo `propietario.types.ps1xml`.

Le permitiremos descubrirlo, posteriormente le explicaremos elemento por elemento cómo está constituido este último:

```
<?xml version="1.0" encoding="utf-8" ?>

<Types>
  <Type>
    <Name>System.IO.FileInfo</Name>
    <Members>
      <ScriptProperty>
        <Name>Owner</Name>
```

```

        <GetScriptBlock>
            $this.GetAccessControl().getOwner(`
[System.Security.Principal.NTAccount])
        </GetScriptBlock>
    </ScriptProperty>
</Members>
</Type>
</Types>

```

Éste está inspirado en el archivo `types.ps1xml` entregado en el estándar de PowerShell.

Como puede ver, esto sigue siendo relativamente sencillo y comprensible. Además, teniendo en cuenta el servicio que un archivo como éste puede proporcionar, sería un error descartarlo.

Algunas explicaciones sobre su estructura:

La primera línea contiene el encabezado estándar de un archivo XML.

A continuación, el elemento raíz `Types`. Después, por cada nuevo tipo o tipo a extender, deberá crear un elemento `Type`. Indicará a continuación el nombre del tipo contemplado en el elemento `Name` y abrirá una etiqueta `Members`. Esto dará cabida a cada nueva propiedad o método personalizado. Para definir una propiedad, utilice el elemento `ScriptProperty`, y para un método `ScriptMethod`. Llega a continuación el nombre de la propiedad, después «la inteligencia» de éste en un elemento `GetScriptBlock`.

Puede ver que en un bloque de código utilizamos la variable `$this` para hacer referencia al objeto y de este modo acceder a sus propiedades y métodos.

a. Utilización de la propiedad Owner

Acabamos de definir la propiedad `owner`. Para verificarla, nada más sencillo que utilizar el comando siguiente para que PowerShell cargue el nuevo archivo de definición del tipo: `Update-TypeData propietario.types.ps1xml`.



Tenga cuidado, no obstante, con la política que escoja para la ejecución del script (véase el capítulo La seguridad). Los archivos `*.ps1xml` son archivos de descripción, pero están firmados digitalmente. Tenga cuidado también con el posible mensaje de error referente al nombre de la firma de este tipo de archivo en el momento de cargarlo con el comando `Update-TypeData`.

Ahora, si utiliza el comando `Get-Member` para obtener la lista de las propiedades, debería ver aparecer el `owner`.

```

PS > Get-Item miArchivo.txt | Get-Member -Type ScriptProperty

TypeName: System.IO.FileInfo

Name      MemberType      Definition
----      -
Owner     ScriptProperty  System.Object Owner {get=$this.GetAccessContr...
Mode      ScriptProperty  System.Object Mode {get=$catr = "";...

```

Para verificar nuestra nueva propiedad, pruebe lo siguiente:

```

PS > (Get-Item miArchivo.txt).Owner

Value
-----
wrkoscar\Oscar

```

O bien, en otra forma:

```
PS > Get-ChildItem *.txt | Format-Table Name,Owner -autosize
```

Name	Owner
-----	-----
datos.txt	wrkoscar \Oscar
MiArchivo.txt	wrkoscar \Armando
test.txt	BUILTIN\Administradores

b. Adición de la segunda propiedad OwnerSID

Si hubiésemos querido añadir una segunda propiedad, por ejemplo la propiedad `ownersid`, habría sido necesario añadir otro elemento `scriptProperty` del mismo modo que anteriormente. Como a continuación:

```
<?xml version="1.0" encoding="utf-8" ?>

<Types>
  <Type>
    <Name>System.IO.FileInfo</Name>
    <Members>
      <ScriptProperty>
        <Name>Owner</Name>
        <GetScriptBlock>
          $this.GetAccessControl().getOwner(`
[System.Security.Principal.NTAccount])
        </GetScriptBlock>
      </ScriptProperty>

      <ScriptProperty>
        <Name>OwnerSID</Name>
        <GetScriptBlock>
          $this.GetAccessControl().getOwner(`
[System.Security.Principal.SecurityIdentifier])
        </GetScriptBlock>
      </ScriptProperty>
    </Members>
  </Type>
</Types>
```

Al igual que en el ejemplo anterior, no olvide cargar su archivo de tipo con el commandlet `Update-TypeData`.

Para verificar su nueva propiedad, pruebe los siguiente:

```
PS > (Get-Item miArchivo.txt).OwnerSID | Format-List

BinaryLength      : 28
AccountDomainSid  : S-1-5-21-2069618812-4153402021-1334178849
Value              : S-1-5-21-2069618812-4153402021-1334178849-1002
```



Puede, a su elección, crear un sólo archivo de tipos personalizados y poner todas sus extensiones en el interior (creando un nuevo elemento Tipo al mismo nivel que el existente para cada nuevo tipo a extender), o bien crear un archivo (*.types.ps1xml) por tipo a extender.

c. Adición de los métodos personalizados SetOwner y GetMSDNHelp

Vayamos un poco más lejos con nuestro ejemplo añadiendo dos métodos:

- **SetOwner**: nos permitirá cambiar el propietario de un archivo.
- **GetMSDNHelp**: gracias a él vamos a poder solicitar ayuda sobre el tipo de objeto durante su utilización. Este método nos abrirá directamente el sitio de Internet de MSDN.



Este ejemplo está extraído del «Blog de Janel» (véase el Capítulo Recursos complementarios - Recursos Web externos).

Para implementar el método `setOwner`, añada el fragmento de código siguiente al conjunto de elementos `scriptProperty` del ejemplo anterior.

```
<ScriptMethod>
  <Name>SetOwner</Name>
  <Script>
    $argument = $args[0]
    $a = $this.GetAccessControl()
    $a.SetOwner([System.Security.Principal.NTAccount]$argument)
    $this.SetAccessControl($a)
  </Script>
</ScriptMethod>
```

Como puede suponer, `setOwner` necesita que se le pase un argumento de entrada para funcionar.

Para utilizarlo, haga lo siguiente:

```
PS > (Get-Item miArchivo.txt).SetOwner('miDominio\miUsuario')
```



Para añadir un método, vamos a utilizar el elemento `scriptMethod` en lugar de `scriptProperty` que sirve para añadir una propiedad. Al igual que en el interior de la definición de un método, será necesario utilizar el elemento `script` en lugar de `getScriptBlock` para una propiedad.

d. Aplicación del método **GetMSDNHelp**

Para verificar este método, necesitaremos crear un nuevo archivo `*.types.ps1xml`, al que llamaremos, por ejemplo, `MSDN.types.ps1xml`.

```
<?xml version="1.0" encoding="utf-8" ?>
<Types>
  <Type>
    <Name>System.Object</Name>
    <Members>
      <ScriptMethod>
        <Name>GetMSDNHelp</Name>
        <Script>
          $culture = $host.currentculture
          if ($args[0]) { $culture = $args[0] }
          if (($global:MSDNViewer -eq $null) -or
              ($global:MSDNViewer.HWND -eq $null))
          {
            $global:MSDNViewer =
              new-object -ComObject InternetExplorer.Application
          }
        </Script>
      </ScriptMethod>
    </Members>
  </Type>
</Types>
```

```

    }
    $Uri = 'http://msdn.microsoft.com/' + $culture `
        + '/library/' + $this.GetType().FullName + '.ASPX'
    $global:MSDNViewer.Navigate2($Uri)
    $global:MSDNViewer.Visible = $TRUE
    $ShellObj = new-object -com WScript.Shell
    $ShellObj.AppActivate((get-process |
        where {$_.MainWindowHandle -eq $global:MSDNViewer.HWND}).Id)
</Script>
</ScriptMethod>
</Members>
</Type>
</Types>

```

Ahora, como hacemos habitualmente, utilizaremos el comando: `Update-TypeData MSDN.types.ps1xml`

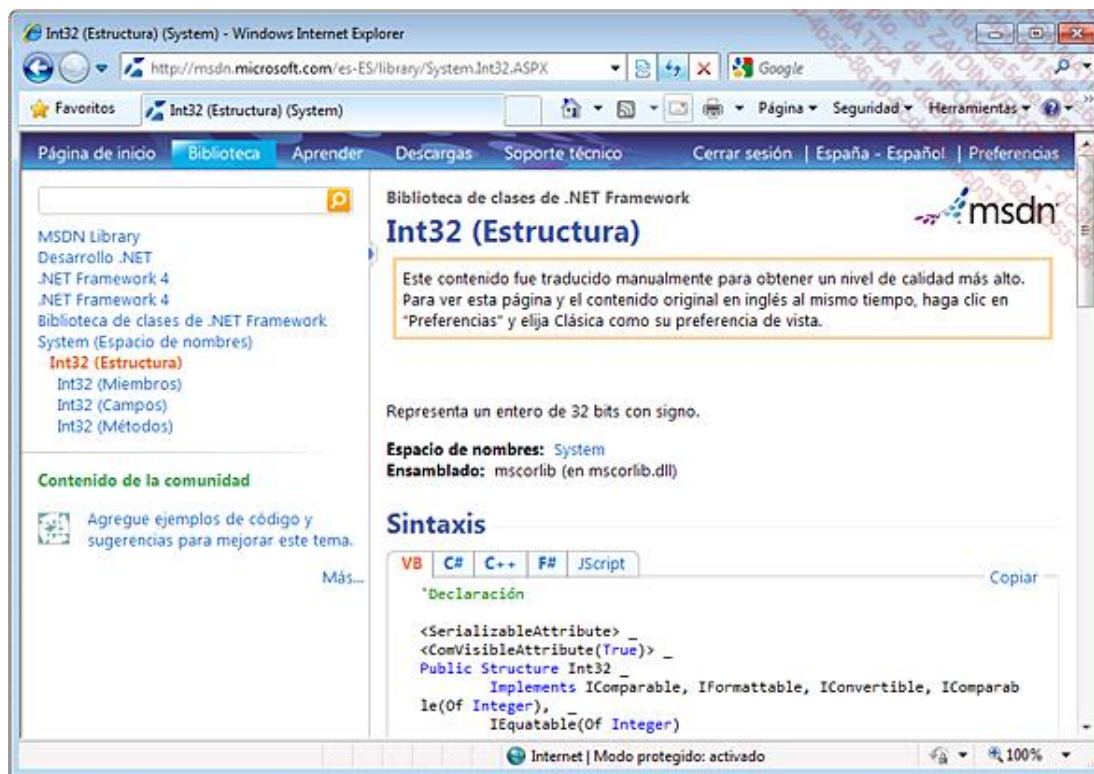
■ Probemos nuestro nuevo método:

```

PS > [int]$var = 66
PS > $var.GetMSDNHelp()

```

Nuestro método funciona: Internet Explorer se abre el sitio MSDN y nos da la información del tipo «Int32» de nuestra variable `$var`.



Verificación del método `GetMSDNHelp`



Para obtener información más detallada acerca de la extensión de tipos, puede dirigirse a la siguiente dirección: <http://msdn2.microsoft.com/en-us/library/ms714665.aspx>