Objetos PSBase y PSObject

Hablemos ahora de una información muy poco documentada, pero muy útil, que son los objetos PSBase y PSObject. Como le habíamos señalado, PowerShell se basa en el Framework .NET. De este modo, los objetos que manipulamos, son un su gran mayoría objetos .NET.

Pero PowerShell también está preparado para funcionar con otros objetos como los objetos COM y WMI, que no comparten la misma tecnología. De hecho, cuando usamos estos otros objetos, PowerShell nos ofrece una representación común, con propiedades y métodos. Es en cierto modo una capa de abstracción, para armonizar la interfaz, independiente de la tecnología del objeto. Para ello, PowerShell utiliza lo que se denomina una adaptación del tipo («Type Adaptation») realizada por PSObjet. Este objeto va a hacer de «Wrapping» (que proviene de WRAP que significa envolver) del objeto de base. Esta adaptación de tipo transforma a su vez el objeto y lo viste añadiéndole algunos métodos nativos de PSObjet y que por tanto nos los encontramos por todas partes, como *ToString*, *CompareTo*, *Equals*, etc.

Tomemos como ejemplo el caso de un objeto de tipo *DateTime*:

```
PS > $Date = Get-Date
```

Veamos ahora toda la información en relación con este objeto tal como PowerShell nos lo presenta con una adaptación de tipo PSObject. Para ello, tecleraremos la línea siguiente:

```
PS > $date.PsObject

Members             : {DisplayHint, DateTime, Date, Day...}
Properties          : {DisplayHint, DateTime, Date, Day...}
Methods             : {Add, AddDays, AddHours, AddMilliseconds...}
ImmediateBaseObject : 10/12/2008 08:00:00
BaseObject          : 10/12/2008 08:00:00
TypeNames           : {System.DateTime, System.ValueType, System.Object}
```

Se observa que existen muchas propiedades para describir cada una de las informaciones sobre el objeto. El detalle de estas propiedades lo representamos en la tabla siguiente:

Propiedad	Descripción
Member	Lista todos los miembros del objeto. Esto incluye a los miembros del objeto de base, los miembros extendidos, y los miembros nativos de un objetoPSObject.
Properties	Lista todos las propiedades del objeto. Incluyendo tanto las propiedades del objeto de base como las propiedades extendidas.
Methods	Lista todos los métodos del objeto. Incluyendo tanto los métodos del objeto de base como los métodos extendidos.
ImmediateBaseObject	Devuelve el objeto de base encapsulado por PSObject.
BaseObject	Devuelve el objeto de base.
TypeName	Lista el nombre de los tipos del objeto.

Utilizando únicamente esta vista que nos proporciona PowerShell, a veces nos privamos de alguna de las funcionalidades del objeto de tecnología subyacente. Y esto puede plantear algunos problemas. Tomemos el ejemplo presentado en el capítulo Manipulación de objetos de directorio con ADSI, que consiste en listar los grupos de una base de cuentas local.

Empecemos por tanto por crear una conexión a la base SAM (*Security Account Manager*) gracias al comando siguiente:

```
PS > $conexion = [ADSI]'WinNT://.'
```

Luego miramos qué métodos vamos a poder aplicar al objeto devuelto por la propiedad Children:

```
PS > $child = $conexion.Children

PS > $child | Get-Member


   TypeName: System.Management.Automation.PSMethod

Name                      MemberType   Definition
----                      ----------   ----------
Copy                      Method       System.Management.Automation
Equals                    Method       System.Boolean Equals(Object obj)
GetHashCode               Method       System.Int32 GetHashCode()
GetType                   Method       System.Type GetType()
get_IsInstance            Method       System.Boolean get_IsInstance()
get_MemberType            Method       System.Management.Automation
get_Name                  Method       System.String get_Name()
get_OverloadDefinitions Method       System.Collections.ObjectModel....
get_TypeNameOfValue     Method       System.String get_TypeNameOfValue()
get_Value                 Method       System.Object get_Value()
```

Nos llevaremos una sorpresa, ya que los miembros listados no son los previstos. La cuestión es ¿cómo podemos acceder a esas funciones del objeto enmascaradas? Pues bien, es aquí donde interviene PSBase. Este último le ofrece una vista sobre el objeto de base (de origen), y no sobre la interfaz PowerShell del objeto.

Realizando la misma operación, pero esta vez utilizando la propiedad *Children* del objeto de base, obtenemos lo siguiente:

```
PS > $child = $conexion.PSBase.Children
PS > $child | Get-Member

   TypeName: System.DirectoryServices.DirectoryEntry

Name                        MemberType    Definition
----                        ----------    ----------
AutoUnlockInterval          Property      System.Directory
BadPasswordAttempts         Property      System.Directory
Descripción                 Property      System.Directory
FullName                    Property      System.Directory
HomeDirDrive                Property      System.Directory
HomeDirectory               Property      System.Directory
LastLogin                   Property      System.Directory
LockoutObservationInterval Property      System.Directory
LoginHours                  Property      System.Directory
LoginScript                 Property      System.Directory
MaxBadPasswordsAllowed      Property      System.Directory
MaxPasswordAge              Property      System.Directory
MaxStorage                  Property      System.Directory
MinPasswordAge              Property      System.Directory
MinPasswordLength           Property      System.Directory
Name                        Property      System.Directory
objectSid                   Property      System.Directory
Parameters                  Property      System.Directory
PasswordAge                 Property      System.Directory
```

```
PasswordExpired            Property      System.Directory
PasswordHistoryLength      Property      System.Directory
PrimaryGroupID             Property      System.Directory
Profile                    Property      System.Directory
UserFlags                  Property      System.Directory
```

Los miembros son totalmente diferentes a los presentados nativamente por PowerShell. Así, utilizando dichas propiedades, hemos accedido a las del objeto de base, y podemos continuar nuestro script.

```
# Get-LocalGroups.ps1

param ([String]$maquina='.')
$conexion = [ADSI]'WinNT://$maquina'
$conexion.PSBase.children |
Where {$_.PSBase.SchemaClassName -eq 'group'} | Foreach{$_.Name}
```