

# Manipular los objetos .NET

La utilización de los objetos .NET abre las puertas a PowerShell a miles de clases listas para utilizar. Por consiguiente, manipular los objetos .NET, facilita una mayor flexibilidad a los scripts y también eleva el nivel hasta flirtear con la programación de objetos.

Para ilustrar estas propuestas, vamos a explicar etapa por etapa cómo aprovechar algunos objetos a través de distintas situaciones que un administrador del sistema puede encontrar, como:

- el envío de un correo,
- el Wake On Lan (encendido remoto),
- la gestión del registro de eventos de puestos remotos,
- la compresión de archivos.

## 1. Enviar un e-mail

En este ejemplo, vamos a detallar el envío de un e-mail en base a los objetos propuestos por el Framework. Ante todo, veamos qué clases están disponibles en el espacio de nombres System.Net.Mail. Para ello, le invitamos a listarlos gracias a la función `Get-TypeName` creada anteriormente.

```
PS > Get-TypeName System.Net.Mail

System.Net.Mail.AttachmentBase
System.Net.Mail.AlternateView
System.Net.Mail.AlternateViewCollection
System.Net.Mail.Attachment
System.Net.Mail.AttachmentCollection
System.Net.Mail.LinkedResource
System.Net.Mail.LinkedResourceCollection
System.Net.Mail.MailAddress
System.Net.Mail.MailAddressCollection
System.Net.Mail.DeliveryNotificationOptions
System.Net.Mail.MailMessage
System.Net.Mail.MailPriority
System.Net.Mail.SendCompletedEventHandler
System.Net.Mail.SmtpDeliveryMethod
System.Net.Mail.SmtpClient
System.Net.Mail.SmtpException
System.Net.Mail.SmtpFailedRecipientException
System.Net.Mail.SmtpFailedRecipientsException
System.Net.Mail.SmtpAccess
System.Net.Mail.SmtpPermissionAttribute
System.Net.Mail.SmtpPermission
System.Net.Mail.SmtpStatusCode
```

Son una veintena de clases en total. Aunque los nombres sean bastante explícitos, no vendrá de más dar un vistazo al sitio web de MSDN para conocer la descripción.

Para el envío de correo «clásico», es decir sin acuse de recibo ni adjuntos, nos interesaremos únicamente por las categorías «MailMessage» (clase que representa un correo electrónico) y «SmtpClient» (clase que permite el envío de correos electrónicos con la ayuda del protocolo SMTP (*Simple Mail Transfer Protocol*)).

La primera etapa consiste en crear nuestro objeto «message» de tipo `System.Net.Mail.MailMessage`. Para ello utilizamos el comando siguiente:

```
PS > $Message = New-Object System.Net.Mail.MailMessage
```

Una vez creado el objeto «message», a continuación será necesario configurarlo, es decir definirle un cierto número de atributos como el remitente, el destinatario, el asunto y el cuerpo del mensaje. Para ello, veamos las propiedades disponibles del objeto con el comando `Get-Member`:

```
PS > $Message | Get-Member
```

En nuestro ejemplo, nos limitaremos a atribuir a nuestro objeto los parámetros esenciales:

```
$message.Body = 'Mensaje enviado con PowerShell'
$message.From = 'robot@powershell-scripting.com'
$message.Subject = 'Mi primer mensaje'
$message.To.Add('admin@powershell-scripting.com')
```

En este punto, sólo nos queda una sola cosa por hacer: enviar el mensaje mediante el protocolo SMTP. Y para ello bastará con tres comandos.

El primero para crear un objeto de tipo `SmtpClient`:

```
PS > $client = New-Object System.Net.Mail.SmtpClient
```

El segundo para conectar el cliente al servidor SMTP que va a permitir el envío del correo:

```
PS > $client.Set_host('SERVIDOR2008')
```

Y el último para enviar definitivamente el mensaje:

```
PS > $client.Send($message)
```

Veamos la versión completa en forma de script:

Script: Envío de un e-mail

```
#Send-email.ps1
#Script que permite el envio de un e-mail

$remitente = 'remitente@host.com'
$destinatario = 'destinatario@host.com'
$servidor = 'mail.host.com'
$asunto = 'Envío de correo vía powershell' + $(get-date)
$texto = 'este es el cuerpo del mensaje'

# Creación del objeto MailMessage
$message = New-Object System.Net.Mail.MailMessage


# Adición de las propiedades
$message.Body = $texto
$message.From = $remitente
$message.Subject = $asunto
$message.To.Add($destinatario)

# Creación del objeto SmtpClient
```

```
$client = New-Object System.Net.Mail.SmtpClient


# Definición de la propiedad del servidor
$client.Set_Host($servidor)

# Envío del mensaje con el método Send
$client.Send($message)
```

 Este ejemplo del envío de correo hace referencia a la versión 1 PowerShell. En efecto, PowerShell v2 integra nativamente el comando Send-MailMessage y este es mucho más completo que nuestro ejemplo.

## 2. Wake On Lan

El Wake On Lan (WOL) es un proceso que permite encender un puesto apagado mediante el envío, a través de la red Ethernet, de un conjunto de bytes un poco especial denominado «paquete mágico».

 Hoy día, prácticamente todas las placas base lo soportan, no obstante puede que el Wake On LAN esté desactivado por defecto en la BIOS.

El paquete mágico que permite desencadenar el WOL es un conjunto de 102 bytes donde los 6 primeros adoptan el valor hexadecimal FF, y los 96 siguientes son 16 veces la repetición de la dirección MAC (*Media Access Control*) de la tarjeta de red del ordenador remoto. Para crear este paquete, utilizaremos la tabla de bytes siguiente:

```
PS > [byte[]]$Direccion_Mac = 0x00, 0x11, 0x43, 0x0E, 0x97, 0x4F
PS > [byte[]]$paquete = 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
PS > $paquete += $Direccion_Mac * 16
```


Un vez constituido el paquete, se ha de enviar a través de la red. Y para ello, vamos a utilizar la clase `UdpClient` (en el espacio de nombres `System.Net.Sockets`) que nos provee de los servicios de red necesarios para el envío de datagramas UDP (*User Datagram Protocol*):

```
PS > $UdpClient = New-Object System.Net.Socket.UdpClient
```

Es gracias a esta clase y más particularmente al método `Connect` que se va a poder establecer una conexión con una máquina remota. Basta después una simple llamada al método `Send` para finalizar el envío del datagrama:

```
PS > $UdpClient.Connect(([System.Net.IPAddress]::Broadcast),1600)
PS > $UdpClient.Send($Paquete,$Paquete.Length)
```

Tenga en cuenta que en la llamada al método `Connect`, utilizamos también la propiedad estática *Broadcast* que devuelve la dirección IP de broadcast (255.255.255.255) para garantizar una difusión general del datagrama, así como el número de puerto 1600.

 En el envío de un datagrama, tenga cuidado de no escoger un puerto ya utilizado por otra aplicación. Recuerde que los puertos utilizados por defecto son los puertos del 0 al 1023.

He aquí nuestro script de Wake On Lan completo:

*Script: Script de Wake On Lan*

```
# WOL.ps1
# Script que permite encender una máquina remota
```

```
[byte[]]$Direccion_Mac = 0x00, 0x11, 0x43, 0x0E, 0x97, 0x4F
[byte[]]$paquete = 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
$paquete += $Direccion_Mac * 16
$UdpClient = New-Object System.Net.Sockets.UdpClient
$UdpClient.Connect(([System.Net.IPAddress]::Broadcast),1600)
$UdpClient.Send($Paquete,$Paquete.Length)
```

En consecuencia, si el puesto remoto está conectado a la red, y si su placa base está correctamente configurada para realizar el WOL, tendrá la agradable sorpresa de despertar un ordenador en pleno sueño.

### 3. Gestionar el registro de eventos

Pasamos ahora a la gestión del registro de eventos. En su configuración básica, PowerShell integra un commandlet llamado Get-EventLog que permite obtener la información referente al registro de eventos del ordenador local.

```
PS > Get-EventLog -list
```

Max(K)	Retain	OverflowAction	Entries	Name
10	240	0 OverwriteAsNeeded	23	Antivirus
20	480	0 OverwriteAsNeeded	1 008	Application
20	480	0 OverwriteAsNeeded	6 903	System
15	360	0 OverwriteAsNeeded	242	Windows PowerShell

Sin embargo, este commandlet no permite el acceso a la información contenida en el registro de eventos de puestos remotos. Y para solucionar lo que puede parecer a una carencia, vamos a hacer una llamada a la clase System.Diagnostics.EventLog del Framework .NET ya que ésta posee un método que permite acceder a los registros de una máquina remota.



Los eventos de tipo seguridad sólo son accesibles con una cuenta con privilegios.

El método es en este caso el método estático GetEventLogs.

```
PS > $Eventos = [System.Diagnostics.EventLog]::GetEventLogs('SERVIDOR2008')
```

```
PS > $Eventos
```

Max(K)	Retain	OverflowAction	Entries	Name
20.480	0	OverwriteAsNeeded	2.749	Application
20.480	0	OverwriteAsNeeded	0	HardwareEvents
512	7	OverwriteOlder	0	Internet Explorer
20.480	0	OverwriteAsNeeded	0	Key Management Service
8.192	0	OverwriteAsNeeded	0	Media Center
16.384	0	OverwriteAsNeeded	0	ODiag
16.384	0	OverwriteAsNeeded	41	OSession
20.480	0	OverwriteAsNeeded	2.886	Security
20.480	0	OverwriteAsNeeded	12.312	System
15.360	0	OverwriteAsNeeded	467	Windows PowerShell

Este método devuelve una tabla de objetos, donde cada elemento corresponde a un registro de eventos particular. Y para conocer el contenido de un registro, utilizaremos la propiedad « entries ».

```
PS > $Eventos[0].Entries
```


Index	Time	Type	Source	EventID	Message
-----	----	----	-----	-----	-----
1	sept. 15 2...	Info	ESENT	100	svchost (632) Windows: El motor ...
2	sept. 15 2...	Info	ESENT	101	svchost (632) Windows: El motor ...
3	sept. 15 2...	Info	ESENT	100	svchost (632) Windows: El motor ...

La utilización del registro de eventos vía el Framework .NET nos permite recuperar todos los eventos de un puesto remoto. Queda ahora determinar qué elementos guardar y cuales no.

Para ayudarle en esta tarea de clasificación de eventos, el ejemplo siguiente le muestra cómo determinar en una línea de comandos, todos los eventos del registro que corresponden a un error de aplicación (ID evento 1000).

```
PS > [System.Diagnostics.EventLog]::GetEventLogs('SERVIDOR2008') |
Where-Object {$_.Get_LogDisplayName() -eq 'Application'} |
Foreach($_.entries) | Where-Object{$_.EventID -eq 1000}
```

Index	Time	Type	Source	EventID	Message
-----	----	----	-----	-----	-----
43	sept. 10 2...	Info	LoadPerf	1000	La descripción del ID de evento '1073742824'
61	sept. 10 2...	Info	LoadPerf	1000	La descripción del ID de evento '1073742824'
140	sept. 10 2...	Info	LoadPerf	1000	La descripción del ID de evento '1073742824'

 El ejemplo anterior que hemos propuesto no es el único medio de acceder al registro de eventos de un puesto remoto. Las consultas WMI también permiten obtener el mismo resultado e incluso optimizar las peticiones en un gran número potencial de eventos.

## 4. Comprimir un archivo

Pasemos a otro ejemplo. Consideremos ahora cómo comprimir y descomprimir un archivo sin pasar por instrumentos de terceros. Esta funcionalidad, que hizo su aparición en el Framework 2.0, utiliza clases situadas en el espacio de nombres System.IO.Compression. Y la clase que va a centrar especialmente nuestra atención es la clase `GZipStream` que proporciona los métodos y las propiedades que permiten la compresión y la descompresión de flujos de datos.

Sin embargo, observando más de cerca el constructor de esta clase, se advierte que necesita, no un archivo, sino un flujo (Stream).

Para proporcionarle el flujo que necesita, es necesario por lo tanto utilizar la clase `FileStream` disponible en el espacio de nombre System.IO.

Para comprimir un archivo, la primera etapa consiste en recuperar el flujo de información del archivo (de hecho, su contenido en forma de un conjunto de bytes) y copiarlo en un tabla de bytes. Para ello la primera acción consiste en instanciar la clase `FileStream` teniendo por parámetro constructor la ruta del fichero, así como la forma en que el sistema operativo debe abrir el archivo:

```
PS> $Stream = New-Object System.IO.Filestream <Ruta_del_Archivo>, 'open'
```

Después llega el momento de la creación de un buffer de tamaño suficiente para incluir el flujo de información:

```
PS > $buffer = New-Object System.Byte[] $Stream.length
```

Aplicamos el contenido del objeto `$Stream` del primer al último byte del flujo al buffer gracias al método `Read`. Y liberamos el archivo:

```
PS > $Stream.Read($buffer,0,$Stream.length)
PS > $Stream.Close()
```

Una vez recuperamos el flujo, debemos comprimirlo gracias a la clase `GZipStream`, e insertarlo en el archivo. Esto implica en primer lugar la creación de un flujo, siempre con el objeto `System.IO.FileStream`:

```
PS > $Stream = New-Object System.IO.FileStream <Nombre_del_Archivo>, 'create'
```

Luego procedemos a la creación de un objeto de tipo «flujo comprimido». Para ello, utilizaremos aquí la clase `GZipStream` con el argumento `$Stream` para especificar en qué flujo (`FileStream`) queremos insertar la información, y `Compress` para elegir un flujo de tipo comprimido.

```
PS > $archivo_gzip = New-Object System.IO.Compression.
GZipStream($Stream, 'compress')
```

Falta escribir la totalidad del flujo comprimido con el método `Write` y luego liberar el archivo.

```
PS > $archivo_gzip.Write($buffer,0,$buffer.Length)
PS > $archivo_gzip.Close()
```

Ahora que tenemos todos los elementos de la compresión a mano, creamos una función que será interesante mantener en su perfil.

```
function Convert-ToGzip {

param([string]$archivo)
if(Test-Path $archivo)
{
    $Stream = New-Object System.IO.FileStream $archivo, 'open'
    $buffer = New-Object System.Byte[] $Stream.length
    $Stream.Read($buffer,0,$Stream.length)
    $Stream.Close()
    $nombre_zip=$archivo + '.Gzip'
    $Stream = New-Object System.IO.FileStream $nombre_zip, 'create'
    $archivozip =
    New-Object System.IO.Compression.GZipStream($Stream,'compress',0)
    $archivozip.Write($buffer,0,$buffer.Length)
    $archivozip.Close()
    Write-Host 'Fin de la compresión'
}
}
```

Veamos ahora qué resultado podemos esperar de una compresión como esta. Para saberlo, creamos un archivo de texto con el resultado del comando `Get-Process`, y utilizamos nuestra función `ConvertTo-Gzip` para obtener igualmente una versión comprimida:

```
PS > Get-Process | Out-File 'c:\Scripts\Process.txt'
PS > Convert-ToGzip -archivo 'c:\Scripts\Process.txt'
```

Finalmente, observamos el resultado con el comando `Get-ChildItem`:

```
PS > Get-ChildItem
```

Directorio: C:\Scripts

Mode	LastWriteTime	Length	Name
----	-----	-----	----
-a---	23/09/2007 11:48	12598	Process.txt
-a---	23/09/2007 11:50	2151	Process.Gzip

El resultado será inapelable, 12598 bytes contra 2151 tras la compresión.