

Monitorización de recursos con la administración de eventos

Otra faceta de WMI bastante desconocida pero muy útil es la administración de los eventos (o *events* en inglés). WMI nos permite supervisar o monitorizar los eventos enviándonos una notificación. A continuación deja en nuestras manos decidir qué acción emprender sobre la recepción de tal o cual evento.

La administración de los eventos WMI puede resultar un formidable aliado de los administradores de sistemas, y así evitar que se conviertan en auténticos apaga fuegos. En efecto, gracias a este mecanismo, vamos, por ejemplo, a estar prevenidos en caso de llenarse un 80 % de un disco lógico de una máquina. «Estar prevenidos» puede significar recibir un e-mail, un SMS, o un pop-up; lo que dependerá únicamente de su script y del modo de notificación que haya escogido. Como hemos comentado, podemos ser notificados de la llegada de un evento sobre cualquier recurso gestionado por WMI. Por tanto, podemos controlar el correcto funcionamiento de determinados servicios en algunos ordenadores remotos, supervisar la ejecución de un proceso en particular, o incluso monitorizar algunas clave de registro. Vistas todas las clases WMI disponibles, existen muchas posibilidades para que pueda monitorizar EL recurso que necesitaba y con esto le permitirá en el futuro poder dormir a pierna suelta...

Para aquellos de ustedes que conozcan *System Center Operations Manager* (SCOM, ex MOM), sabrán que el principio de notificaciones de eventos es el mismo; y para los que no lo conozcan sepan que podrán hacer lo mismo que SCOM pero a una escala bastante menor...

Si no existiesen las notificaciones WMI, para tratar de hacer lo mismo, estaríamos obligados a desarrollar scripts que se ejecutarían a intervalos de tiempo regulares para poder controlar ciertos recursos administrados. Aunque esto sea posible, esta técnica podría llegar a consumir una gran cantidad de recursos puesto que el script debería ejecutarse muy a menudo. Las notificaciones WMI están precisamente pensadas para ser la vía más eficiente para realizar estas tareas.

1. Monitorizar la creación de un proceso

Tomemos un ejemplo sencillo: la supervisión del proceso MSPaint.exe correspondiente a la aplicación de diseño estándar de Windows que todo el mundo conoce. El objetivo es desencadenar un evento cuando el sistema detecte el inicio del proceso MSPaint.exe. El evento será simplemente la visualización de un mensaje.

Utilizando la versión 1 de PowerShell escribiremos el script siguiente:

```
# Monitorización del proceso mspaint.exe

$strComputer = '.'
$query = "SELECT * FROM __InstanceCreationEvent
        WITHIN 3
        WHERE TargetInstance ISA 'Win32_process'
        AND TargetInstance.Name='mspaint.exe' "

$query = New-Object System.Management.WqlEventQuery $query

$scope = New-Object `
System.Management.ManagementScope "\\$strComputer\root\cimv2"

$watcher = New-Object `
System.Management.ManagementEventWatcher $scope,$query

$watcher.Start()
$event=$watcher.WaitForNextEvent()
Write-host "Una instancia de MSPaint acaba de crearse."
```


Cuando ejecute este script, verá que la consola PowerShell se bloquea. Esto es así ya que se queda a la espera del proceso MSPaint.exe. La ejecución del script está en cierto modo suspendida a nivel de la petición del método `WaitForNextEvent`. Por tanto cuando se inicie el proceso MSPaint.exe y se detecte por WMI, el script continuará su

ejecución. El comando `Write-Host` visualiza su mensaje, acto seguido el script finaliza.

No utilizamos `Get-WmiObject` en este ejemplo, por la sencilla razón de que este commandlet no tiene en cuenta los eventos WMI. Por tanto, haremos directamente llamadas a las clases .NET disponibles en el espacio de nombres `System.Management`. Crearemos dos objetos: una petición WQL, y un ámbito que indica la estructura WMI en la cual se encuentra la instancia a monitorizar. A partir de estos dos objetos, crearemos un tercero que será un «observador». Éste recibirá el evento cuando se detecte.

Pero ocupémonos unos instantes de la petición WQL ya que es la parte más importante de este ejemplo; y será ésta la única que modificaremos en el futuro para los ejemplos siguientes.

El inicio de la petición es similar a las que ya hemos realizado anteriormente, es decir que está constituida por **SELECT** y **FROM**. Por el contrario, indicamos que queremos obtener los eventos de tipo **__InstanceCreationEvent**; en otras palabras, como su nombre indica, los eventos de creación de objetos. En segundo lugar, aparecen dos nuevas palabras clave: **WITHIN** y **ISA**. La primera indica un intervalo en segundos que determinará la frecuencia de ejecución del administrador de eventos, y el segundo indica que la instancia a monitorizar debe pertenecer a una determinada clase WMI. Por último, se define el nombre de la instancia en la que centraremos nuestra atención con **TargetInstance.Name='proceso'**. Si no hubiéramos indicado un nombre de proceso («MSPaint.exe»), el script nos habría devuelto la primera instancia de proceso detectada.

 Todos los scripts dados en ejemplo pueden ejecutarse sin problema en una máquina remota (en la medida en que usted tenga los privilegios adecuados) simplemente sustituyendo el contenido de la variable `strComputer` por un nombre de ordenador o una dirección IP.

Aunque este ejemplo sea plenamente operativo con PowerShell v2, sin embargo, podremos introducirle algunas modificaciones para aprovechar las nuevas funcionalidades introducidas por la nueva versión.

PowerShell v2 dispone del comando `Register-WmiEvent`. El cual permite suscribirse a los eventos WMI; estos últimos en la terminología WMI se denominan «suscriptores».

`Register-WmiEvent` posee los parámetros siguientes:

Parámetro	Descripción
<code>Action <ScriptBlock></code>	Especifica comandos que controlan los eventos. Los comandos del parámetro <code>Action</code> se ejecutan cuando se genera un evento, en lugar de enviar el evento a la cola de eventos. Incluya los comandos entre llaves (<code>{ }</code>) para crear un bloque de script. El valor del parámetro <code>Action</code> puede incluir las variables automáticas <code>\$Event</code> , <code>\$EventSubscriber</code> , <code>\$Sender</code> , <code>\$SourceEventArgs</code> y <code>\$SourceArgs</code> , que proporcionan información acerca del evento al bloque de script <code>Action</code> .
<code>Class <String></code>	Especifica el evento al que se está suscribiendo. Escriba la clase de WMI que genera los eventos. Se requiere un parámetro <code>Class</code> o <code>Query</code> en cada comando.
<code>ComputerName <String></code>	Especifica un equipo remoto. El valor predeterminado es el equipo local. Escriba un nombre NetBIOS, una dirección IP o un nombre de dominio completo.
<code>Credential <PSCredential></code>	Especifica una cuenta de usuario con permiso para realizar esta acción. Escriba un nombre de usuario, como "Usuario01" o "Dominio01\Usuario01". O bien, escriba un objeto <code>PSCredential</code> , como el devuelto por el cmdlet <code>Get-Credential</code> . Si escribe un nombre de usuario se le pedirá una contraseña.
<code>Forward [<SwitchParameter>]</code>	Envía los eventos de esta suscripción a la sesión en el equipo local. Use este parámetro si se registra para eventos en un equipo remoto o en una sesión remota.
<code>MessageData <PSObject></code>	Especifica los datos adicionales que se van a asociar a esta suscripción de

	eventos. El valor de este parámetro aparece en la propiedad <code>MessageData</code> de todos los eventos asociados a esta suscripción.
<code>Namespace <String></code>	Especifica el espacio de nombres de la clase de WMI.
<code>Query <String></code>	Especifica una consulta en Lenguaje de consulta de WMI (WQL) que identifica la clase de eventos de WMI, como "select * from __InstanceDeletionEvent".
<code>SourceIdentifier <String></code>	<p>Especifica un nombre que se selecciona para la suscripción. El nombre que se seleccione debe ser único en la sesión actual. El valor predeterminado es el GUID asignado por Windows PowerShell.</p> <p>El valor de este parámetro aparece en el valor de la propiedad <code>SourceIdentifier</code> del objeto de suscriptor y de todos los objetos de evento asociados a esta suscripción.</p>
<code>SupportEvent [<SwitchParameter>]</code>	<p>Oculto la suscripción de eventos. Use este parámetro si la suscripción actual forma parte de un mecanismo de registro de eventos más complejo y no debe detectarse de manera independiente.</p> <p>Para ver o cancelar una suscripción creada con el parámetro <code>SupportEvent</code>, use el parámetro <code>Force</code> de los cmdlets <code>Get-EventSubscriber</code> y <code>Unregister-Event</code>.</p>
<code>Timeout <Int64></code>	<p>Determina el tiempo que Windows PowerShell espera para que este comando se complete.</p> <p>El valor predeterminado, 0 (cero), significa que no hay tiempo de espera, y hace que Windows PowerShell espere de forma indefinida.</p>

Ahora que sabemos más sobre este commandlet, veamos cómo podemos reescribir nuestro anterior script:

```
#Requires -Version 2
# Monitorización del proceso MSPaint.exe - v2

$query = "SELECT * FROM __InstanceCreationEvent
        WITHIN 3
        WHERE TargetInstance ISA 'Win32_process'
        AND TargetInstance.Name='mspaint.exe' "

$action = {Write-Host "
Una instancia de MSPaint acaba de crearse con fecha $($event.timegenerated)."}

Register-WmiEvent -query $query -SourceId 'Paint' -Action $action
```

Lo que salta inmediatamente a la vista por comparación con la versión anterior de nuestro script es su menor tamaño. Pero abstrayéndose de estas consideraciones puramente estéticas podremos señalar los siguientes aspectos:

- La petición WMI es siempre la misma.
- `Register-WmiEvent` ha dado considerablemente más claridad al script; lo que facilita por tanto su comprensión.
- La acción a efectuar en la aparición del evento está claramente identificada por un bloque de script.
- Hemos utilizado la variable `$event`. Ésta contiene la referencia del objeto correspondiente al evento que ha sido desencadenado. De este modo recuperaremos la fecha y la hora de activación.

Iniciemos el script con el fin de ver como funciona:

```
PS > ./monitoring_PaintProcess.ps1
```

Id	Name	State	HasMoreData	Location	Command
--	----	-----	-----	-----	-----
8	Paint	NotStarted	False		Write-Host "..."

El script se lleva a cabo inmediatamente al contrario que en la versión anterior. Esto se debe al hecho de que en PowerShell v2 la administración de los eventos WMI se realiza por el mecanismo de jobs en segundo plano (véase el capítulo Control del Shell). El resultado de la ejecución del comando `Get-Job` será el mismo que en el caso anterior, salvo que nos hubiera mostrado todos los jobs en curso de ejecución.

Observe bien el campo «*State*». Podemos ver que nuestro evento llamado «*Paint*» está en estado «*NotStarted*», dicho de otro modo, no se ha iniciado todavía.

Si ahora iniciamos la aplicación MS Paint, ésto es lo que nos devuelve la consola PowerShell:

Una instancia de MSPaint acaba de crearse con fecha 10/22/2009 23:28:34.

Ahora que hemos iniciado el evento, veamos de nuevo el estado de este último con el comando `Get-Job`:

```
PS > Get-Job
```

Id	Name	State	HasMoreData	Location	Command
--	----	-----	-----	-----	-----
8	Paint	Running	True		Write-Host "..."

Podemos constatar que ahora el estado ha pasado a «*Running*» y que la propiedad `HasMoreData` ha pasado del valor «*False*» a «*True*».

Observe que igualmente podemos utilizar el commandlet `Get-EventSubscriber` para obtener la información complementaria sobre nuestro administrador de eventos WMI:

```
PS > Get-EventSubscriber
```

```
SubscriptionId      : 8
SourceObject        : System.Management.ManagementEventWatcher
EventName           : EventArrived
SourceIdentifier     : Paint
Action              : System.Management.Automation.PSEventJob
HandlerDelegate     :
SupportEvent        : False
ForwardEvent        : False
```

Si desea «desregistrar» el administrador de eventos con el fin, por ejemplo, de liberar memoria, podemos hacer lo siguiente:

```
PS > Get-EventSubscriber -SubscriptionId 8 | Unregister-Event
```

O

```
PS > Get-Job -id 8 | Remove-Job -force
```

El parámetro `-force` es necesario para eliminar los jobs que no se han ejecutado todavía.

Observe que si no se indican los parámetros `-subscriptionId` y `-id`, se eliminarán todos los jobs.

2. Monitorizar la tasa de ocupación de disco de un servidor

Como le decíamos al comienzo de este apartado, podemos supervisar el espacio de un disco de una máquina local o remota de la red. Con WMI, bastará indicar el nombre de la máquina remota en lugar del valor «.». En los scripts basados en WMI es una buena práctica utilizar una variable para indicar el nombre de la máquina en la que el script se aplica. En nuestros scripts, utilizamos para este fin, la variable `$StrComputer`.

En este ejemplo, procederemos a desencadenar una notificación cuando el porcentaje de espacio libre del disco C: sea inferior a 10 GB. Basaremos nuestra petición en la clase `win32_LogicalDisk`. Nos limitaremos a mostrar un mensaje en pantalla indicando que se ha alcanzado un umbral crítico y mostraremos el espacio de disco restante. Observe que no es lo mejor que se puede hacer «en la vida real», pero evidentemente está en nuestras manos escribir un script más evolucionado. Como PowerShell v2 proporciona facilidades para el envío de correos, modificaremos la segunda versión de nuestro script con el fin de notificar al administrador del sistema mediante el envío de un e-mail.

Con la versión 1 de PowerShell podemos construir el script de la siguiente forma:

```
# Monitorización del espacio de disco restante en C:

$StrComputer = '.'

$query = "SELECT * FROM __InstanceModificationEvent
        WITHIN 3
        WHERE TargetInstance ISA 'Win32_LogicalDisk'
        AND TargetInstance.DeviceID = `\"C:`"
        AND TargetInstance.FreeSpace < 10737418240"

$query = New-Object System.Management.WqlEventQuery $query

$scope = New-Object System.Management.ManagementScope `
"\$StrComputer\root\cimv2"
$watcher = New-Object System.Management.ManagementEventWatcher `
$scope,$query
$watcher.Start()

While ($true)
{
    $w=$watcher.WaitForNextEvent()
    $freeSpace = $w.TargetInstance.FreeSpace/1GB
    $freeSpace = [System.Math]::Round($freeSpace,2)
    Write-Host "¡Umbral crítico alcanzado!
    Espacio restante: $FreeSpace Gb"
}
```

El resultado de la ejecución de este script podría ser el siguiente:

```
¡Umbral crítico alcanzado!
    Tamaño restante: 8.63 Gb
```

Verá que esta vez, puesto que buscamos los eventos de tipo modificación, hemos reemplazado `__InstanceCreationEvent` por `__InstanceModificationEvent`. Hemos modificado el intervalo de monitorización considerando que no era útil hacer una solicitud a la máquina cada tres segundos, ya que cada sesenta bastaba.

Los más atentos habrán observado también una pequeña sutileza en la asignación de la propiedad **TargetInstance.DeviceID** a nivel de la petición WQL. En efecto, hemos tenido que emplear el carácter de escape «backtick» (```) delante de las comillas. Ha sido necesario ya que la petición debe contener obligatoriamente comillas, y si no utilizamos los backticks, PowerShell considerará que cerramos a la cadena de caracteres abierta anteriormente. Lo que provocaría un error. Hemos utilizado igualmente el backtick en las líneas siguientes con el fin de «partir» las líneas de script demasiado largas para hacer de este modo nuestro script más comprensible a los lectores.

Hemos utilizado también un método estático del Framework .NET, el de la clase matemática (Math) denominado Round. De este modo redondearemos el resultado de la conversión Bytes -> GigaBytes a dos decimales.

Finalmente, gracias a la instrucción **While** creamos un bucle infinito con el fin de que el script continúe indefinidamente su ejecución una vez reciba la notificación.

Este ejemplo, todo y ser totalmente funcional con PowerShell v2, puede sin embargo mejorarse con el fin de sacar partido de las nuevas posibilidades aportadas por esta versión.

He aquí la segunda versión de nuestro script:

```
#Requires -Version 2
# Monitorización del espacio libre en disco C:

$query = "SELECT * FROM __InstanceModificationEvent
        WITHIN 3
        WHERE TargetInstance ISA 'Win32_LogicalDisk'
        AND TargetInstance.DeviceID = `\"C:\""
        AND TargetInstance.FreeSpace < 10737418240"

$action = {
    $e = $Event.SourceEventArgs.NewEvent
    $freeSpace = $e.TargetInstance.FreeSpace/1GB
    $freeSpace = [System.Math]::Round($freeSpace,2)
    Write-Host ";Umbral crítico alcanzado!
    Espacio restante: $FreeSpace Gb"
}

register-wmiEvent -query $query -sourceid 'EspacioLibre' -action $action
```

Veamos el resultado después de iniciarlo:

Id	Name	State	HasMoreData	Location	Command
--	----	-----	-----	-----	-----
5	EspacioLibre	NotStarted	False		...

Dicho y hecho. Vamos a enviar un e-mail en lugar de mostrar una cadena de caracteres, lo que reflejará mejor la realidad...

PowerShell v2 nos aporta el commandlet Send-MailMessage, todo y que no detallaremos aquí su funcionamiento. Veamos ahora nuestro script modificado:

```
#Requires -Version 2
# Monitorización del espacio de disco restante en C:

$query = "SELECT * FROM __InstanceModificationEvent
        WITHIN 3
        WHERE TargetInstance ISA 'Win32_LogicalDisk'
        AND TargetInstance.DeviceID = `\"C:\""
        AND TargetInstance.FreeSpace < 10737418240"

$action = {
    $e = $Event.SourceEventArgs.NewEvent
    $freeSpace = $e.TargetInstance.FreeSpace/1GB
    $freeSpace = [System.Math]::Round($freeSpace,2)
    $message = ";Umbral crítico alcanzado! Espacio restante: $FreeSpace Gb"
```

```

    Send-MailMessage -to 'admin@misociedad.es' -from 'robot@misociedad.es' `
        -subject 'Espacio de disco bajo' -body $message -smtpServer
    mailsrv.misociedad.es
}

register-wmiEvent -query $query -sourceid 'EspacioLibre' -action $action

```

3. Monitorizar la eliminación de archivos

Después de haber ilustrado la monitorización de la creación y de la modificación de instancias, no nos queda más que probar la eliminación de instancias. En este ejemplo, seguiremos de cerca el directorio «c:\temp» con el objetivo de detectar cualquier eliminación de archivos en el interior del mismo.

```

# Monitorización de la eliminación de archivos en C:\temp - v1

$strComputer = '.'
$query = "SELECT * FROM __InstanceDeletionEvent
    WITHIN 3
    WHERE TargetInstance ISA 'CIM_DirectoryContainsFile'
    AND TargetInstance.GroupComponent='Win32_Directory.Name='\"C:\\\\temp`\"'"

$query = New-Object System.Management.WqlEventQuery $query

$scope = New-Object `
    System.Management.ManagementScope "\\$strComputer\root\cimv2"

$watcher = New-Object `
    System.Management.ManagementEventWatcher $scope,$query
$watcher.Start()

$file=$watcher.WaitForNextEvent()
Write-Host "¡Archivo $($file.TargetInstance.PartComponent) eliminado!"

```

El resultado de la ejecución de este script sería algo como:

```

¡Archivo \\WIN7_ESCRITORIO\root\cimv2:CIM_DataFile.Name="C:\\temp\\test.txt"
eliminado!

```

La ruta corresponde a la ruta del archivo en formato WMI; a continuación es tarea nuestra extraer el nombre del archivo.

Con PowerShell v2, podemos transformar este script de la manera siguiente:

```

#Requires -Version 2
# Monitorización de la eliminación de archivos en C:\temp - v2

$query = "SELECT * FROM __InstanceDeletionEvent
    WITHIN 3
    WHERE TargetInstance ISA 'CIM_DirectoryContainsFile'
    AND TargetInstance.GroupComponent='Win32_Directory.Name='\"C:\\\\temp`\"'"

$action =
{
    $e = $Event.SourceEventArgs.NewEvent
    Write-Host "¡El archivo $($e.TargetInstance.PartComponent) ha sido

```

```
eliminado!"
```

```
}
```

```
Register-WmiEvent -query $query -sourceid 'eliminacion' -action $action
```

4. Algunas explicaciones complementarias

Habr  podido observar a lo largo de los diferentes ejemplos que  nicamente cambia la petici n WQL; el resto del script es pr cticamente id ntico. Un buen dominio de la administraci n de los eventos WMI pasa necesariamente por una buena comprensi n del lenguaje WQL y sobre todo del esquema b sico WMI.

Hemos utilizado las tres clases de eventos siguientes:

- __InstanceCreationEvent
- __InstanceModificationEvent
- __InstanceDeletionEvent

Estas clases nos han permitido monitorizar las instancias, o en otras palabras, los objetos de nuestro sistema operativo. Sepa que existen otras categor as para monitorizar las operaciones en las clases y en los espacios de nombres WMI pero  stas presentan poco inter s para la mayor a de administradores de sistema. Finalmente otra categor a de clases de eventos susceptible de centrar nuestra atenci n es la que permite el seguimiento del registro de Windows; pero no vamos a insistir mucho en el ya que se basa en el mismo principio.

Evitar salir bruscamente de la consola en la espera de eventos

Cuando hacemos un bucle infinito con `while ($true) { ... }` como en el ejemplo donde monitoriz bamos la tasa de ocupaci n de disco, la  nica forma de interrumpir el script es pulsando **[Ctrl][Pausa]** ya que **[Ctrl] C** no tiene ning n efecto. Aunque esta secuencia de teclas es eficaz, a veces lo es demasiado, ya que cierra tambi n la consola PowerShell.

El problema no radica en la instrucci n `while`, sino en el observador WMI (clase `ManagementEventWatcher`) que espera una notificaci n. En efecto,  ste es insensible a **[Ctrl] C**. Existe de todos modos un peque o truco para saltarnos este problema. Consiste en definir un timeout para nuestro observador WMI. Es decir al cabo de un tiempo de espera, si no se ha recibido una notificaci n, el observador cesa su trabajo y devuelve el control al script. El script dejar  de estar paralizado y la ejecuci n podr  seguir su curso normal. Es por tanto en este momento, cuando podremos realizar un **[Ctrl] C** para salir del script, justo antes que se ponga nuevamente a la espera de otra notificaci n. Sin embargo, para que esto funcione correctamente, tendremos que efectuar un «trap» de error ya que el observador, al t rmino del timeout, emite una excepci n. Y si no «atrapamos» esta excepci n, el script se interrumpir  ya que se trata de un error cr tico (si lo cree oportuno consulte el cap tulo Control del Shell acerca de la administraci n de los errores). Definiremos por tanto un administrador de interceptaci n, quien, cuando atrape una excepci n de tipo **System.Management.ManagementException** se asegurar  de que el script contin a normalmente su ejecuci n. Tamb n haremos una prueba con el fin de determinar si un evento se ha producido o no. Y si se da el caso, entonces mostraremos un mensaje.

He aqu  nuestro segundo ejemplo revisado y corregido:

```
# Monitorizaci n del espacio de disco restante en C: - v1
# con la posibilidad de salir con [CTRL]+C

$strComputer = '.'
$query = "SELECT * FROM __InstanceModificationEvent
        WITHIN 3
        WHERE TargetInstance ISA 'Win32_LogicalDisk'
```



```

        AND TargetInstance.DeviceID = `"C:"`
        AND TargetInstance.FreeSpace < 10737418240"
$query = New-Object System.Management.WqlEventQuery $query

$scope = New-Object System.Management.ManagementScope `
"$\$strComputer\root\cimv2"

$watcher = New-Object `
System.Management.ManagementEventWatcher $scope,$query
$options = New-Object System.Management.EventWatcherOptions
$options.Timeout = [timespan]"0.0:0:1"           # Timeout de 1 segundo
$watcher.Options = $options
$watcher.Start()

While ($true){
    trap [System.Management.ManagementException] {continue}
    $w=$watcher.WaitForNextEvent()

    if ($w.TargetInstance -ne $null) {
        $freeSpace = $w.TargetInstance.FreeSpace/1GB
        $freeSpace = [System.Math]::Round($freeSpace,2)

        Write-Host ";Umbral crítico alcanzado!
            Espacio restante: $freeSpace Gb"

        $w = $null
    }
}

```

Utilizando esta técnica, podemos decir que estamos realizando una administración de los eventos semi síncrona.

- Si queremos evitar la visualización en la consola del mensaje de error provocado por la excepción (incluso si el script continúa su ejecución), debemos dar el valor **SilentlyContinue** a la variable de preferencia **\$ErrorActionPreference**. Si no lo hacemos, este será el mensaje de error que obtendremos: Excepción por la llamada de «WaitForNextEvent» con «0» argumento(s): «Plazo excedido».

Pero con la Version 2 de PowerShell todo este trabajo se vuelve superfluo ya que, de base, si la administración de los eventos se realiza con Register-WmiEvent es asíncrona.

Para los usuarios de PowerShell v1:

- La administración de los eventos asíncronos, a diferencia de los eventos síncronos, no tiene por objeto bloquear la ejecución de un script. En este contexto, los eventos asíncronos deberían en teoría ser detectados en tarea de fondo. Su administración con PowerShell depende de la programación avanzada más que del scripting. Esta la razón por la que el proyecto open-source «PowerShell Eventing» se ha creado en el sitio web comunitario CodePlex.com, y unió un pequeño grupo de desarrolladores durante aproximadamente cinco meses. Es así como ha visto la luz un pequeño conjunto de comandos dedicados a la administración de eventos (síncronos y asíncronos). Como información, estos commandlets con nombres evocadores son: **New-Event**, **Get-Event**, **Get-EventBinding**, **Connect-EventListener**, **Disconnect-EventListener** y **Start-KeyHandler**.

- Descárguese el snap-in PowerShell Eventing de la dirección siguiente: <http://www.codeplex.com/PSEventing>