Es

Creating HTML Reports in PowerShell

Don Jones
Principal Author

Creating HTML Reports in PowerShell (Spanish)

The DevOps Collective, Inc.

Este libro está a la venta en
http://leanpub.com/creating-html-reports-in-powershell-spanish

Esta versión se publicó en 2018-10-28

También por The DevOps Collective, Inc.

Creating HTML Reports in Windows PowerShell

A Unix Person's Guide to PowerShell

The Big Book of PowerShell Error Handling

DevOps: The Ops Perspective

Ditch Excel: Making Historical and Trend Reports in PowerShell

Secrets of PowerShell Remoting

The Big Book of PowerShell Gotchas

The Monad Manifesto, Annotated

Why PowerShell?

Windows PowerShell Networking Guide

The PowerShell + DevOps Global Summit Manual for Summiteers

Why PowerShell? (Spanish)

Secrets of PowerShell Remoting (Spanish)

DevOps: The Ops Perspective (Spanish)

The Monad Manifesto: Annotated (Spanish)

The Big Book of PowerShell Gotchas (Spanish)

The Big Book of PowerShell Error Handling (Spanish)

DevOps: WTF?

PowerShell.org: History of a Community

Índice general

Creating HTML Reports in PowerShell

Por Don Jones

Aprenda a utilizar correctamente ConvertTo-HTML para producir informes HTML de varias secciones y bien formados, pero luego vaya más allá con un módulo EnhancedHTML personalizado. Produaca informes hermosos, codificados por colores, dinámicos y con multi-secciones de forma fácil y rápida. Escrito por Don Jones.

Descargar el código El módulo EnhancedHTML2 mencionado en este libro puede encontrarse en PowerShell Gallery[1]. Esa página incluye instrucciones de descarga. PowerShellGet es necesario, y se puede obtener de PowerShellGallery.com

¿Ha sido útil este libro? El (los) autor (es) le pide (n) que haga una donación deducible de impuestos (en los EE.UU., consulte sus leyes si vive en otro lugar) de cualquier cantidad a The DevOps Collective[2] para apoyar su trabajo.

[1] https://www.powershellgallery.com/packages/EnhancedHTML2/
[2] https://devopscollective.org/donate/

Revise las actualizaciones! Nuestros ebooks se actualizan a menudo con contenido nuevo y corregido. Los hacemos disponibles de tres maneras::

- Nuestra rama principal GitHub organization[3], con un repositorio para cada libro. Visite https://github.com/devops-collective-inc/
- Nuestra GitBook page[4], donde puede navegar por los libros en línea, o descargarlos en formato PDF, EPUB o MOBI. Utilizando el lector en línea, puede saltar a capítulos específicos. Visite https://www.gitbook.com/@devopscollective
- En LeanPub[5], donde se pueden descargar como PDF, EPUB, o MOBI (login requerido), y "comprar" los libros haciendo una donación a DevOps. También puede elegir recibir notificaciones de actualizaciones. Visite https://leanpub.com/u/devopscollective

GitBook y LeanPub generan la salida del formato PDF ligeramente diferente, por lo que puede elegir el que prefiera. LeanPub también le puede notificar cada vez que liberamos alguna actualización. Nuestro repositorio de GitHub es el principal; los repositorios en otros sitios suelen ser sólo espejos utilizados para el proceso de publicación. GitBook normalmente contendrá nuestra última versión, incluyendo algunos bits no terminados; LeanPub siempre contiene la más reciente "publicación liberada" de cualquier libro.

[3] https://github.com/devops-collective-inc
[4] https://www.gitbook.com/@devopscollective
[5] https://leanpub.com/u/devopscollective

Bases del informe HTML

En primer lugar, entender que PowerShell no se limita a crear informes en HTML. Pero me gusta el HTML porque es flexible, puede ser enviado fácilmente a través de correo electrónico, y se ve mucho mejor que un simple informe de texto sin ningún formato. Pero antes de sumergirnos en esto, necesitamos aclarar un poco cómo funciona el HTML.

Una página HTML es sólo un archivo de texto sin formato, con algo parecido a esto:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" \
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>HTML TABLE</title>
</head><body>
<table>
<colgroup><col/><col/><col/><col/><col/></colgroup>
<tr><th>ComputerName</th><th>Drive</th><th>Free(GB)</th><\
th>Free(%)</th><th>Size(GB)</th></tr>
<tr><td>CLIENT</td><td>C:</td><td>49</td><td>82</td><td>6\
0</td></tr>
</table>
</body></html>
```

Cuando se interpreta por un navegador, este archivo se representa en la pantalla que aparece en la ventana del navegador. Lo mismo se aplica a los clientes de correo electrónico capaces de mostrar contenido HTML. Mientras que usted, como persona, puede poner obviamente cualquier cosa en el archivo, necesita seguir las reglas que los browsers esperan para obtener la salida deseada.

Una de esas reglas es que cada archivo debe contener uno y un solo documento HTML. Es todo el contenido entre la etiqueta `<HTML>` y la etiqueta `</HTML>` (los nombres de las etiquetas no distinguen entre mayúsculas y minúsculas, y es común verlas en minúsculas como en el ejemplo anterior). Menciono esto porque una de las cosas más comunes que veré a la gente hacer con PowerShell se parecerá a esto:

```
Get-WmiObject -class Win32_OperatingSystem | ConvertTo-HT\
ML | Out-File report.html
Get-WmiObject -class Win32_BIOS | ConvertTo-HTML | Out-Fi\
le report.html -append
Get-WmiObject -class Win32_Service | ConvertTo-HTML | Out\
-File report.html -append
```

"Aaarrrggh," dice mi colon cada vez que veo eso. Básicamente, está diciendo a PowerShell que cree tres documentos HTML completos y los coloque en un solo archivo. Mientras que algunos navegadores (Internet Explorer, por ejemplo) entenderán eso e intentarán mostrar algo, es simplemente incorrecto hacer esto. Una vez que empiece generar esta clase de informes, descubrirá rápidamente que este enfoque es doloroso. No es culpa de PowerShell; Simplemente no está siguiendo las reglas. ¡Por eso esta guía!

Se dará cuenta que el HTML consiste en muchas otras etiquetas, como: `<TABLE>`, `<TD>`, `<HEAD>`, y otras más. La mayoría de estas forman parejas, lo que significa que vienen en una etiqueta de apertura como `<TD>` y una etiqueta de cierre como `</TD>`. La etiqueta `<TD>` representa una celda de tabla, y todo entre esas etiquetas se considera el contenido de esa celda.

La sección `<HEAD>` es importante. Lo que hay dentro no es normalmente visible en el navegador. En su lugar, el navegador se centra en lo que hay en la sección `<BODY>`. La sección `<HEAD>` proporciona metadatos adicionales, como el título de la página (lo se muestra en la barra de título o pestaña de la ventana del navegador, no en

la página), las hojas de estilo o las secuencias de comandos que se adjuntan a la página, y cosas así. Vamos a hacer algunas cosas impresionantes con la sección `<HEAD>`, confíe en mí.

También notará que este HTML es bastante "limpio", en contraposición, digamos, a la salida HTML de Microsoft Word. Este HTML no contiene información visual incrustada en él, como colores o fuentes. Eso es bueno, porque sigue las buenas prácticas de HTML de separar la información de formato de la estructura del documento. Será decepcionante al principio, porque sus páginas HTML parecerán algo aburridas. Pero vamos a mejorar eso, también.

Para ayudar a que la narrativa de este libro permanezca enfocada, voy a comenzar con un solo ejemplo. En ese ejemplo, vamos a recuperar varios bits de información acerca de una computadora remota y formatear todo en un bonito y dinámico informe HTML. Con suerte, podrá concentrarse en las técnicas que estoy mostrando, y adaptarlas a sus propias necesidades.

En mi ejemplo, quiero que el informe tenga cinco secciones, cada una con la siguiente información:

- Información de la computadora
- Versión del sistema operativo del equipo, número de compilación y versión del Service Pack.
- Información de hardware: la cantidad de RAM instalada y el número de cores, junto con el fabricante y el modelo.
- Una lista de todos los procesos que se ejecutan en la máquina.
- Una lista de todos los servicios que están configurados para iniciarse automáticamente, pero que no se están ejecutando.
- Información sobre todos los adaptadores de red físicos en el equipo. No direcciones IP, necesariamente - información de hardware como la dirección MAC

Soy consciente que esta información no es un conjunto universalmente interesante, pero estas secciones permitirán demostrar

algunas técnicas específicas. Una vez más, espero que usted pueda adaptar esto a sus necesidades precisas.

Recopilación de la información

Soy un gran fan de la programación modular. Gran, gran fan. Con eso en mente, tiendo a escribir funciones que recopilan la información que quiero incluir en mi informe, y normalmente haré una función por sección principal de mi informe. Verá dentro de poco cómo es eso de beneficioso. Escribiendo cada función individualmente, hago más fácil de usar esa misma información en otras tareas, y hago más fácil depurar cada una. El truco consiste en que cada salida de función sea un solo tipo de objeto que combine toda la información de esa sección para el informe. He creado cinco funciones, que he pegado en un solo archivo de script. Le mostraré cada una de esas funciones una a la vez, con un breve comentario. Aquí va la primera:

```
function Get-InfoOS {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)][string]$ComputerName
    )
    $os = Get-WmiObject -class Win32_OperatingSystem -Com\
puterName $ComputerName
    $props = @{'OSVersion'=$os.version;
               'SPVersion'=$os.servicepackmajorversion;
               'OSBuild'=$os.buildnumber}
    New-Object -TypeName PSObject -Property $props
}
```

Esta es una función sencilla, y la principal razón por la que me molesté en hacer que sea una función - en lugar de simplemente

usar Get-WmiObject directamente - es que quiero nombres de propiedad diferentes, como "OSVersion" en lugar de sólo "Version". Dicho esto, tiendo a seguir exactamente este mismo patrón de programación para todas las funciones de recuperación de información, sólo para mantenerlas consistentes.

```
function Get-InfoCompSystem {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)][string]$ComputerName
    )
    $cs = Get-WmiObject -class Win32_ComputerSystem -Comp\
uterName $ComputerName
    $props = @{'Model'=$cs.model;
               'Manufacturer'=$cs.manufacturer;
               'RAM (GB)'="{0:N2}" -f ($cs.totalphysicalm\
emory / 1GB);
               'Sockets'=$cs.numberofprocessors;
               'Cores'=$cs.numberoflogicalprocessors}
    New-Object -TypeName PSObject -Property $props
}
```

Muy similar a la anterior. Notará aquí que estoy usando el operador de formato -f con la propiedad RAM, de modo que obtengo un valor en gigabytes con 2 decimales. El valor nativo está en bytes, lo que no es útil para mí.

```
function Get-InfoBadService {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)][string]$ComputerName
    )
    $svcs = Get-WmiObject -class Win32_Service -ComputerN\
ame $ComputerName `
           -Filter "StartMode='Auto' AND State<>'Running'"
    foreach ($svc in $svcs) {
        $props = @{'ServiceName'=$svc.name;
                   'LogonAccount'=$svc.startname;
                   'DisplayName'=$svc.displayname}
        New-Object -TypeName PSObject -Property $props
    }
}
```

Aquí, he tenido que reconocer que voy a estar recuperando más de un objeto de WMI, así que tengo que enumerar a través de ellos usando una construcción ForEach. Una vez más, estoy principalmente renombrando propiedades. Podría haber hecho eso con un comando Select-Object, pero me gusta mantener la estructura de funciones general similar a mis otras funciones. Es sólo una preferencia personal que me ayuda a incluir menos errores, ya que estoy acostumbrado a hacer las cosas de esta manera.

```
function Get-InfoProc {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)][string]$ComputerName
    )
    $procs = Get-WmiObject -class Win32_Process -Computer\
Name $ComputerName
    foreach ($proc in $procs) {
        $props = @{'ProcName'=$proc.name;
                   'Executable'=$proc.ExecutablePath}
```

```
        New-Object -TypeName PSObject -Property $props
    }
}
```

Muy similar a la función de servicios. Probablemente pueda empezar a notar cómo usar esta misma estructura hace que una cierta cantidad "copiar y pegar" se vuelva efectiva cuando creo una nueva función.

```
function Get-InfoNIC {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)][string]$ComputerName
    )
    $nics = Get-WmiObject -class Win32_NetworkAdapter -Co\
mputerName $ComputerName `
            -Filter "PhysicalAdapter=True"
    foreach ($nic in $nics) {
        $props = @{'NICName'=$nic.servicename;
                   'Speed'=$nic.speed / 1MB -as [int];
                   'Manufacturer'=$nic.manufacturer;
                   'MACAddress'=$nic.macaddress}
        New-Object -TypeName PSObject -Property $props
    }
}
```

Lo principal para notar aquí es cómo he convertido la propiedad speed, que esta nativamente en bytes, a megabytes. Debido a que no me interesan los decimales aquí (quiero un número entero), eligo utilizar el valor como un entero, utilizando el operador -as, que es más sencillo para mí que el operador de formato -f. ¡Además, me da la oportunidad de mostrar esta técnica!

Tenga en cuenta que, a los efectos de este libro, voy a poner estas funciones en el mismo archivo de script que el resto de mi código, lo que en realidad genera el código HTML. Normalmente no hago eso.

Normalmente, las funciones de recuperación de información van a un módulo de script, y entonces escribo mi script de generación HTML para cargar ese módulo. Tener las funciones en un módulo hace que sean más fáciles de usar en otros lugares, si quiero. Estoy omitiendo el módulo esta vez sólo para mantener las cosas más simples en esta demostración. Si desea obtener más información sobre los módulos de script, debería dar una mirada a Learn PowerShell Toolmaking in a Month of Lunches o PowerShell in Depth, los cuales están disponibles en Manning.com.

Construyendo el HTML

Voy a abandonar el CmdLet nativo de ConvertTo-HTML que he discutido hasta ahora, En lugar de eso, voy a pedirle que utilice el módulo EnhancedHTML2 que viene con este e-Book. Tenga en cuenta que, a partir de octubre de 2013, se trata de una nueva versión del módulo - es más sencillo que el módulo EnhancedHTML introducido con la edición original de este libro.

Comencemos con el script que utiliza el módulo. Se incluye con este libro como EnhancedHTML2-Demo.ps1, por lo que aquí voy a pegarlo aquí y luego agregare las explicaciones sobre lo que hace cada bit. Tenga en cuenta que no puedo controlar cómo se ve el código en un e-Reader, por lo que es probable que parezca un poco desordenado.

```
#requires -module EnhancedHTML2
<#
.SYNOPSIS
Generates an HTML-based system report for one or more com\
puters.
Each computer specified will result in a separate HTML fi\
le;
specify the -Path as a folder where you want the files wr\
itten.
Note that existing files will be overwritten.

.PARAMETER ComputerName
One or more computer names or IP addresses to query.

.PARAMETER Path
The path of the folder where the files should be written.

```

```
18  .PARAMETER CssPath
19  The path and filename of the CSS template to use.
20
21  .EXAMPLE
22  .\New-HTMLSystemReport -ComputerName ONE,TWO `
23                         -Path C:\Reports\
24  #>
25  [CmdletBinding()]
26  param(
27      [Parameter(Mandatory=$True,
28                 ValueFromPipeline=$True,
29                 ValueFromPipelineByPropertyName=$True)]
30      [string[]]$ComputerName,
31
32      [Parameter(Mandatory=$True)]
33      [string]$Path
34  )
```

La sección anterior nos dice que se trata de un "script avanzado", lo que significa que utiliza el enlace de CmdLet de PowerShell. Puede especificar uno o más nombres de equipo para los que se genera el informe, y debe especificar una ruta de acceso de carpeta (no un nombre de archivo) para almacenar los reportes finales.

```
1  BEGIN {
2      Remove-Module EnhancedHTML2
3      Import-Module EnhancedHTML2
4  }
```

El bloque BEGIN podría ser eliminado dependiendo de la versión de PowerShell que esté utilizando. Utilizo esta demostración para probar el módulo, así que es importante que descargue cualquier versión antigua de la memoria (si ha cargado el módulo anteriormente) y vuelva a cargar la versión revisada. De hecho, PowerShell v3 y posterior no requerirá la importación si el módulo está correctamente ubicado en `\Documents\WindowsPowerShell\Modules\EnhancedHTML2`.

```
PROCESS {

$style = @"
<style>
body {
    color:#333333;
    font-family:Calibri,Tahoma;
    font-size: 10pt;
}

h1 {
    text-align:center;
}

h2 {
    border-top:1px solid #666666;
}

th {
    font-weight:bold;
    color:#eeeeee;
    background-color:#333333;
    cursor:pointer;
}

.odd  { background-color:#ffffff; }

.even { background-color:#dddddd; }

.paginate_enabled_next, .paginate_enabled_previous {
    cursor:pointer;
    border:1px solid #222222;
    background-color:#dddddd;
    padding:2px;
    margin:4px;
```

```
36      border-radius:2px;
37  }
38
39  .paginate_disabled_previous, .paginate_disabled_next {
40      color:#666666;
41      cursor:pointer;
42      background-color:#dddddd;
43      padding:2px;
44      margin:4px;
45      border-radius:2px;
46  }
47
48  .dataTables_info { margin-bottom:4px; }
49
50  .sectionheader { cursor:pointer; }
51
52  .sectionheader:hover { color:red; }
53
54  .grid { width:100% }
55
56  .red {
57      color:red;
58      font-weight:bold;
59  }
60  </style>
61  "@
```

Eso se llama hoja de estilos en cascada, o CSS. Hay algunas cosas interesantes para sacar destacar:

He colocado toda la sección `<style></ style>` en una cadena here-string de PowerShell[6], y almacenado en la variable $style. Hará que sea fácil referirse a esto en adelante.

Tenga en cuenta que he definido el estilo de varias etiquetas HTML, como H1, H2, BODY y TH. Esas definiciones de estilo listan el

[6] https://goo.gl/exzNGQ

nombre de la etiqueta sin un signo anterior de período o hash. Se definen los elementos de estilo que interesan, como el tamaño de la fuente, la alineación del texto, etc. Etiquetas como H1 y H2 ya tienen estilos predefinidos establecidos por su navegador, como su tamaño de fuente. Cualquier cosa que ponga en el CSS reemplazará los valores predeterminados del navegador.

Los estilos también heredan. Todo el cuerpo de la página HTML está contenido dentro de las etiquetas `<BODY></ BODY>`, por lo que cualquier cosa que asigne a la etiqueta BODY en CSS también se aplicará a todo lo que contenga la página. Mi cuerpo establece una familia de fuentes y un color de fuente. Las etiquetas H1 y H2 usarán la misma fuente y color.

También verá las definiciones de estilo precedidas por un punto. Esos se llaman estilos de clase. Son clase de plantillas reutilizables del estilo que se pueden aplicar a cualquier elemento dentro de la página. Los ".paginate" son realmente utilizados por el JavaScript que uso para crear tablas dinámicas. No me gustó la forma en que los botones Prev / Next se veían fuera de la caja, así que modifiqué mi CSS para aplicar estilos diferentes.

Preste mucha atención a .odd, .even, y .red en el CSS. Vera que los utilizo poco a poco.

```
function Get-InfoOS {
    [CmdletBinding()]
    param(
        [Parameter(Mandatory=$True)][string]$ComputerName
    )
    $os = Get-WmiObject -class Win32_OperatingSystem -Com\
puterName $ComputerName
    $props = @{'OSVersion'=$os.version
               'SPVersion'=$os.servicepackmajorversion;
               'OSBuild'=$os.buildnumber}
    New-Object -TypeName PSObject -Property $props
}
```

```
13
14  function Get-InfoCompSystem {
15      [CmdletBinding()]
16      param(
17          [Parameter(Mandatory=$True)][string]$ComputerName
18      )
19      $cs = Get-WmiObject -class Win32_ComputerSystem -Comp\
20  uterName $ComputerName
21      $props = @{'Model'=$cs.model;
22                 'Manufacturer'=$cs.manufacturer;
23                 'RAM (GB)'="{0:N2}" -f ($cs.totalphysicalm\
24  emory / 1GB);
25                 'Sockets'=$cs.numberofprocessors;
26                 'Cores'=$cs.numberoflogicalprocessors}
27      New-Object -TypeName PSObject -Property $props
28  }
29
30  function Get-InfoBadService {
31      [CmdletBinding()]
32      param(
33          [Parameter(Mandatory=$True)][string]$ComputerName
34      )
35      $svcs = Get-WmiObject -class Win32_Service -ComputerN\
36  ame $ComputerName `
37              -Filter "StartMode='Auto' AND State<>'Running'"
38      foreach ($svc in $svcs) {
39          $props = @{'ServiceName'=$svc.name;
40                     'LogonAccount'=$svc.startname;
41                     'DisplayName'=$svc.displayname}
42          New-Object -TypeName PSObject -Property $props
43      }
44  }
45
46  function Get-InfoProc {
47      [CmdletBinding()]
```

```
48      param(
49          [Parameter(Mandatory=$True)][string]$ComputerName
50      )
51      $procs = Get-WmiObject -class Win32_Process -Computer\
52  Name $ComputerName
53      foreach ($proc in $procs) {
54          $props = @{'ProcName'=$proc.name;
55                     'Executable'=$proc.ExecutablePath}
56          New-Object -TypeName PSObject -Property $props
57      }
58  }
59
60  function Get-InfoNIC {
61      [CmdletBinding()]
62      param(
63          [Parameter(Mandatory=$True)][string]$ComputerName
64      )
65      $nics = Get-WmiObject -class Win32_NetworkAdapter -Co\
66  mputerName $ComputerName `
67              -Filter "PhysicalAdapter=True"
68      foreach ($nic in $nics) {
69          $props = @{'NICName'=$nic.servicename;
70                     'Speed'=$nic.speed / 1MB -as [int];
71                     'Manufacturer'=$nic.manufacturer;
72                     'MACAddress'=$nic.macaddress}
73          New-Object -TypeName PSObject -Property $props
74      }
75  }
76
77  function Get-InfoDisk {
78      [CmdletBinding()]
79      param(
80          [Parameter(Mandatory=$True)][string]$ComputerName
81      )
82      $drives = Get-WmiObject -class Win32_LogicalDisk -Com\
```

```
puterName $ComputerName `
           -Filter "DriveType=3"
    foreach ($drive in $drives) {
        $props = @{'Drive'=$drive.DeviceID;
                   'Size'=$drive.size / 1GB -as [int];
                   'Free'="{0:N2}" -f ($drive.freespace /\
 1GB);
                   'FreePct'=$drive.freespace / $drive.si\
ze * 100 -as [int]}
        New-Object -TypeName PSObject -Property $props
    }
}
```

Las seis funciones anteriores no hacen otra cosa que recuperar datos de una sola computadora (observe que su parámetro -ComputerName se define como `[string]`, aceptando un valor, en lugar de `[string[]]` que aceptaría múltiples). Si no logra entender cómo funciona esto... es probable que tenga que dar un paso atrás!

Para propósitos de formato, usted está viendo que se utiliza el carácter (back tick) (como en –ComputerName y $ComputerName). En PowerShell este carácter funciona como una especie de continuación de línea. Lo señalo porque puede ser fácil perderlo de vista.

```
foreach ($computer in $computername) {
    try {
        $everything_ok = $true
        Write-Verbose "Checking connectivity to $computer"
        Get-WmiObject -class Win32_BIOS -ComputerName $Co\
mputer -EA Stop | Out-Null
    } catch {
        Write-Warning "$computer failed"
        $everything_ok = $false
    }
```

Lo anterior es el inicio de mi script de demostración. Se están tomando los nombres de equipo que se pasaron al parámetro `-ComputerName`, procesándolos uno a la vez. Luego se hace una llamada a `Get-WmiObject` como una prueba - si esto falla, no quiero hacer nada con el nombre del equipo en absoluto. El resto de la secuencia de comandos sólo se ejecuta si esa llamada WMI tiene éxito.

```
 if ($everything_ok) {
        $filepath = Join-Path -Path $Path -ChildPath "$co\
mputer.html"
```

Recuerde que el otro parámetro de este script es `-Path`. Estoy utilizando `Join-Path` para combinar $Path con un nombre de archivo. Join-Path garantiza el número correcto de barras inversas, de modo que si `-Path` es "C:" o "C:" obtendré una ruta de archivo válida. El nombre de archivo será el nombre del equipo actual, seguido de la extensión .html.

```
        $params = @{'As'='List';
                    'PreContent'='<h2>OS</h2>'}
        $html_os = Get-InfoOS -ComputerName $computer |
                   ConvertTo-EnhancedHTMLFragment @params
```

Aquí está mi primer uso del módulo EnhancedHTML2: Con ConvertTo-EnhancedHTMLFragment. Observe lo que estoy haciendo:

1. Estoy usando un hashtable para definir los parámetros del comando, incluyendo ambos -As List y -PreContent '`<H2>OS</H2>`' como parámetros y sus valores. Esto especifica una salida de estilo de lista (frente a una tabla), precedida por el encabezado "OS" en el estilo H2. Vuelva a mirar el CSS y verá que he aplicado un borde superior a todo el elemento `<H2>`, lo que ayudará a separar visualmente las secciones de mi informe.

2. Estoy ejecutando mi comando Get-InfoOS, pasando el nombre del equipo actual. La salida se canaliza a...
3. ConvertTo-EnhancedHTMLFragment, ConvertTo-EnhancedHTMLFragment, donde se encuentra mi hashtable de parámetros. El resultado será una gran cadena de HTML, que se almacenará en $html_os.

```
        $params = @{'As'='List';
                    'PreContent'='<h2>Computer System</h2\
>'}
        $html_cs = Get-InfoCompSystem -ComputerName $comp\
uter |
                   ConvertTo-EnhancedHTMLFragment @params
```

Ese es un ejemplo similar, para la segunda sección de mi informe..

```
        $params = @{'As'='Table';
                    'PreContent'='<h2>&diams; Local Disks\
</h2>';
                    'EvenRowCssClass'='even';
                    'OddRowCssClass'='odd';
                    'MakeTableDynamic'=$true;
                    'TableCssClass'='grid';
                    'Properties'='Drive',
                @{n='Size(GB)';e={$_.Size}},
                @{n='Free(GB)';e={$_.Free};css={if ($_.Fre\
ePct -lt 80) { 'red' }}},
                @{n='Free(%)';e={$_.FreePct};css={if ($_.F\
reeePct -lt 80) { 'red' }}}}

        $html_dr = Get-InfoDisk -ComputerName $computer |
                   ConvertTo-EnhancedHTMLFragment @params
```

OK, ese es un ejemplo más complejo. Echemos un vistazo a los parámetros que estoy pasando a ConvertTo-EnhancedHTMLFragment:

- Como se está produciendo una tabla en lugar de una lista, la salida será en un diseño de tabla columnar (algo como lo que produciría Format-Table, pero en HTML).
- Para mi sección de encabezado, he añadido un símbolo de diamante utilizando la entidad HTML ♢ Creo que se ve bien. Eso es todo.
- Puesto que esto será una tabla, puedo especificar -EvenRowCssClass y -OddRowCssClass. Le doy los valores "even" y "odd", que son las dos clases (.even y .odd) que definí en mi CSS. De esta forma estoy creando el vínculo entre las filas de la tabla y mi CSS. Cualquier fila de la tabla "etiquetada" con la clase "odd" heredará el formato de ".odd" de mi CSS. No se debe incluir el punto al especificar los nombres de clase con estos parámetros. Sólo en el CSS se coloca el punto delante del nombre de la clase.
- `-MakeTableDynamic` se establece en $True, para que se aplique el JavaScript necesario y convertir la salida en una tabla que se pueda ordenar y paginar. Esto requerirá que el HTML final se vincule al archivo JavaScript necesario, pero cubriremos este punto cuando lleguemos allí.
- `-TableCssClass` es opcional, pero lo estoy usando para asignar la clase "grid". Una vez más, si observa el CSS, podrá observar que definí un estilo para ".grid", por lo que esta tabla heredará esas instrucciones de estilo.
- El último es el parámetro `-Properties`. Funciona muy parecido a los parámetros `-Properties` de `Select-Object` y `Format-Table`. El parámetro acepta una lista de propiedades separada por comas. El primero, Drive, ya está siendo producido por `Get-InfoDisk`. Los siguientes tres son especiales: son hashtables, creando columnas personalizadas como loa haría Format-Tabl. Dentro del hashtable, usted puede utilizar las siguientes claves:
  - n (o name, o l, o label) especifica el encabezado de columna. Estoy usando "Size(GB)," "Free(GB)", y "Free(%)" como encabezados de columna.

- e (o expression) es un bloque de secuencia de comandos, que define lo que contendrá la celda de la tabla. Dentro de ella, puede utilizar $_ para referirse al objeto de entrada. En este ejemplo, el objeto canalizado proviene de `Get-InfoDisk`, por lo que me refiero a las propiedades Size, Free y FreePct del objeto.
- css (o cssClass) es también un bloque de secuencia de comandos. Mientras que el resto de las claves funcionan igual que lo hacen con Select-Object o Format-Table, css (o cssClass) es exclusivo de ConvertTo-EnhancedHTMLFragment. Acepta un bloque de secuencia de comandos, que se espera que produzca una cadena, o nada. En este caso, estoy comprobando para ver si la propiedad FreePct del objeto en la canalización es menor que 80 o no. Si es así, la salida será la cadena "red". Esta cadena se agregará como una clase CSS de la celda en la tabla. Recuerde que en mi CSS definí la clase ".red" y aquí es donde adjunto esa clase a las celdas de la tabla.
- Como una nota aparte, me doy cuenta de que es tonto establecer un color rojo cuando el porcentaje libre de disco es inferior al 80%. Se trata solo de un ejemplo para jugar. Podría fácilmente tener una fórmula más compleja, como *if ($_.FreePct -lt 20) { 'red' } elseif ($_.FreePct -lt 40) { 'yellow' } else { 'green' }* y entonces habría definido las clases ".red", ".yellow" y ".green" en el CSS.

```
$params = @{'As'='Table';
                          'PreContent'='<h2>&diams; Proce\
sses</h2>';
                          'MakeTableDynamic'=$true;
                          'TableCssClass'='grid'}
$html_pr = Get-InfoProc -ComputerName $computer |
                              ConvertTo-EnhancedHTMLFragm\
ent @params

$params = @{'As'='Table';
                          'PreContent'='<h2>&diams; Servi\
ces to Check</h2>';
                          'EvenRowCssClass'='even';
                          'OddRowCssClass'='odd';
                          'MakeHiddenSection'=$true;
                          'TableCssClass'='grid'}

 $html_sv = Get-InfoBadService -ComputerName $computer |
                                ConvertTo-EnhancedHTMLFrag\
ment @params
```

Más de lo mismo en los dos ejemplos anteriores, con sólo un nuevo parámetro: -MakeHiddenSection. Esto hará que la sección del informe se colapse de forma predeterminada, mostrando sólo la cadena -PreContent. Al hacer clic en la cadena, se expandirá y contraerá la sección del informe.

De regreso en mi CSS, observe que para la clase .sectionHeader, establezco el cursor en un icono de puntero, e hice que el color del texto de la sección fuera rojo cuando el ratón pasa sobre él. Esto ayuda a comprender al usuario que se puede hacer clic en el encabezado de la sección. El módulo EnhancedHTML2 siempre agrega la clase CSS “sectionheader” al -PreContent, por lo que al definir “.sectionheader” en su CSS, puede seguir diseñando los encabezados de sección.

```
        $params = @{'As'='Table';
                    'PreContent'='<h2>&diams; NICs</h2>';
                    'EvenRowCssClass'='even';
                    'OddRowCssClass'='odd';
                    'MakeHiddenSection'=$true;
                    'TableCssClass'='grid'}
        $html_na = Get-InfoNIC -ComputerName $Computer |
                   ConvertTo-EnhancedHTMLFragment @params
```

Nada nuevo en el fragmento anterior, pero ahora estamos listos para generar el HTML final:

```
        $params = @{'CssStyleSheet'=$style;
                    'Title'="System Report for $computer";
                    'PreContent'="<h1>System Report for $\
computer</h1>";
            'HTMLFragments'=@($html_os,$html_cs,$html_dr,\
$html_pr,$html_sv,$html_na);
                    'jQueryDataTableUri'='C:\html\jqueryd\
atatable.js';
                    'jQueryUri'='C:\html\jquery.js'}
        ConvertTo-EnhancedHTML @params |
        Out-File -FilePath $filepath

        <#
        $params = @{'CssStyleSheet'=$style;
                    'Title'="System Report for $computer";
                    'PreContent'="<h1>System Report for $\
computer</h1>";
            'HTMLFragments'=@($html_os,$html_cs,$html_dr,\
$html_pr,$html_sv,$html_na)}
        ConvertTo-EnhancedHTML @params |
        Out-File -FilePath $filepath
        #>
    }
```

```
24  }
25
26  }
```

El código no comentado y el código comentado hacen lo mismo. El primero, no comentado, establece una ruta de archivo local para los dos archivos JavaScript necesarios. El comentado no especifica esos parámetros, por lo que el código HTML final utilizará el JavaScript desde la Red de distribución de contenido (CDN) basada en la Web de Microsoft. En ambos casos:

- -CssStyleSheet especifica mi CSS - estoy alimentando mi variable predefinida $style. También puede vincular a una hoja de estilo externa (hay un parámetro diferente, -CssUri, para eso), pero tener el estilo incrustado en el HTML lo hace más autónomo.
- -Title especifica qué se mostrará en la barra de título del navegador o pestaña.
- -PreContent, que estoy definiendo mediante las etiquetas HTML `<H1>`, aparecerá en la parte superior del informe. También hay un -PostContent si desea agregar un pie de página.
- -HTMLFragments requiere una matriz (de ahí el uso de @ () para crear una matriz) de fragmentos HTML producidos por ConvertTo-EnhancedHTMLFragment. Así estoy alimentando las 6 secciones del informe HTML que creé anteriormente.

El resultado final se canaliza a la ruta de archivo que creé anteriormente. Así se ve el resultado:

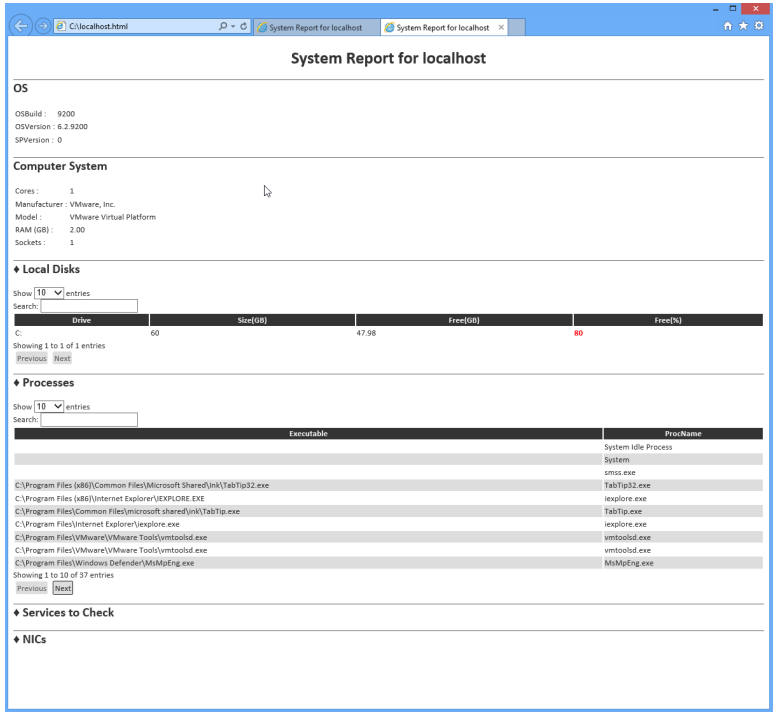

image004.png

Tengo mis últimas dos secciones contraidas. Observe que la lista de procesos está paginada, con los botones Previous/Next y además mi disco sin el 80% está resaltado en rojo. Las tablas muestran 10 filas por defecto, pero se pueden hacer más grandes, y ofrecen un cuadro de búsqueda incorporado. Se puede hacer clic sobre los encabezados de columna para ordenar.

¡Francamente, creo que se ve extraordinario!

Combinación de informes HTML y una aplicación GUI

He tenido un buen número de personas haciendo preguntas en los foros en PowerShell.org, con el asunto "¿cómo puedo utilizar un RichTextBox en una aplicación GUI de Windows para mostrar datos en un formato agradable?" Mi respuesta es no lo haga. Utilice HTML en su lugar. Por ejemplo, digamos que siguió los ejemplos del capítulo anterior y produjo un hermoso informe HTML. Tenga en cuenta que el informe permanece "en memoria", no en un archivo de texto, hasta el final:

```
      $params = @{'CssStyleSheet'=$style;
                  'Title'="System Report for $computer";
                  'PreContent'="<h1>System Report for $\
computer</h1>";
                  'CssIdsToMakeDataTables'=@('tableProc\
','tableNIC','tableSvc');
                  'HTMLFragments'=@($html_os,$html_cs,$\
html_pr,$html_sv,$html_na)}
        ConvertTo-EnhancedHTML @params |
        Out-File -FilePath $filepath
```

Por razones de ilustración, digamos que ahora está en un archivo llamado C:Report.html. Voy a usar PowerShell Studio 2012 de SAPIEN para mostrar ese informe en una GUI, en lugar de hacerla aparecer en un navegador Web. Entonces, he iniciado con un proyecto simple, de una sola forma. He cambiado el texto del formulario a "Informe", y he añadido un control WebBrowser desde

la caja de herramientas. Ese control llena automáticamente la forma entera, así que está perfecto. Nombré el control de WebBrowser "web", lo que hará accesible desde el código a través de la variable $web.

Cabe resaltar que PowerShell Studio 2012 podría estar muy desfasado en el momento que lea esto, pero todavía debería tener la idea general.

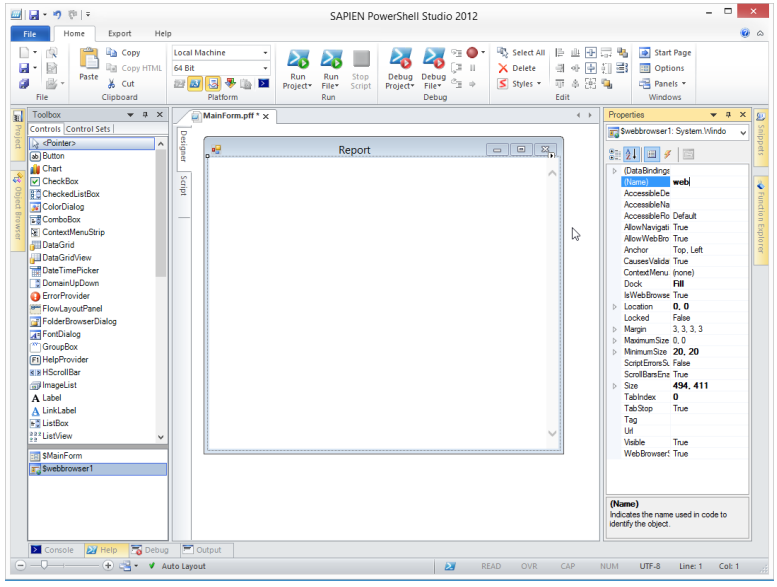

image006.png

Espero que haga un formulario como parte de un proyecto general más grande. Por ahora solo me voy a enfocar en solucionar este problema. Así que voy a cargar la información del reporte en el control WebBrowser cuando el formulario se cargue:

```
$OnLoadFormEvent={
#TODO: Initialize Form Controls here
    $web.Navigate('file://C:\report.html')
}
```

Ahora puedo ejecutar el proyecto:

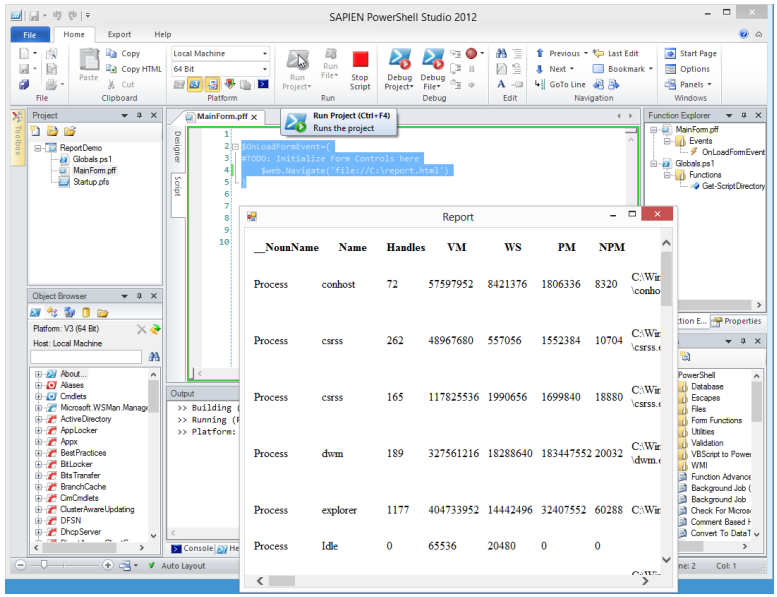

image007.png

Obtengo un agradable cuadro de diálogo emergente que muestra el informe HTML. Puedo cambiar el tamaño, minimizarlo, maximizarlo y cerrarlo usando los botones estándar en la barra de título de la ventana. Fácil, y sólo tomó 5 minutos.

Contactándome

Si tiene problemas, desea hacer algo y no puede averiguar cómo, encontró un error y desea ofrecer una corrección, o simplemente tiene comentarios sobre esta guía o el módulo EnhancedHTML, me encantaría saber de usted.

La forma más sencilla es publicar en el foro "General Q & A" en http://powershell.org/wp/forums/. Mantengo puesto el ojo aquí que yo voy a responder tan pronto como sea posible.

Revise de vez en cuando, para asegurarse que tiene la versión más reciente de esta guía y su código.