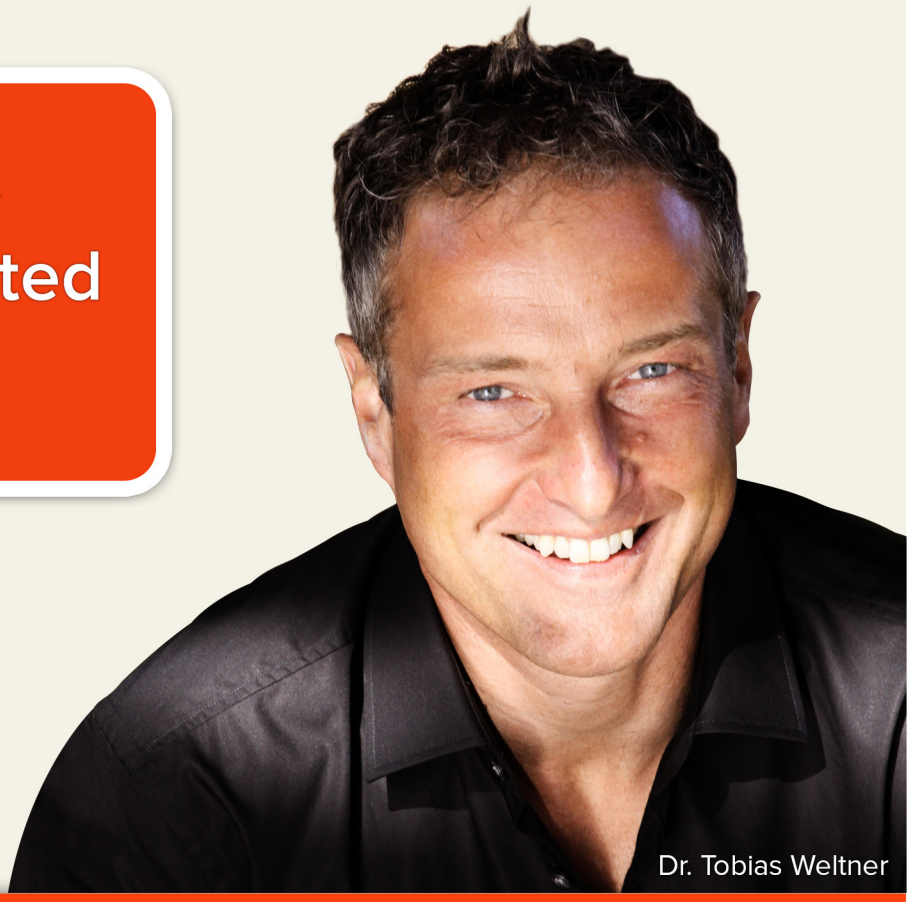
PowerTips MONTHLY

Part of the PowerShell.com reference library, brought to you by **Dr. Tobias Weltner**

Volume 10 March 2014

This Month's Topic:

Internet-Related Tasks



Dr. Tobias Weltner

Table of Contents

1. Downloading Files

To automatically download files from the Internet, you can use a .NET WebClient object. It shares Internet connection settings with Internet Explorer. This will download one of our previous issues of the monthly guide, explaining a little more about objects and .NET types:

```
$url = 'http://powershell.com/cs/media/p/26784/download.aspx'

$object = New-Object Net.WebClient
$localPath = "$home\Documents\objects_and_types.pdf"
$object.DownloadFile($url, $localPath)

explorer.exe "/SELECT,$localPath"
Invoke-Item -Path $localPath
```

2. Get WebClient with Proxy Authentication

If your company is using an Internet proxy, and you'd like to access Internet with a WebClient object, make sure it uses the proxy and supplies your default credentials to it.

You could write a little helper function to get such a pre-configured WebClient:

```
function Get-WebClient
{
    $wc = New-Object Net.WebClient
    $wc.UseDefaultCredentials = $true
    $wc.Proxy.Credentials = $wc.Credentials
    $wc
}
```

Next, combine it with the logic to download things:

```
function Get-WebClient
{
    $wc = New-Object Net.WebClient
    $wc.UseDefaultCredentials = $true
    $wc.Proxy.Credentials = $wc.Credentials
    $wc
}

$url = 'http://powershell.com/cs/media/p/26784/download.aspx'

$object = Get-WebClient
$localPath = "$home\Documents\objects_and_types.pdf"
$object.DownloadFile($url, $localPath)

explorer.exe "/SELECT,$localPath"
Invoke-Item -Path $localPath
```

You could now adjust the function Get-WebClient to your needs. You could extend it to use a proxy server, or supply specific credentials.

3. Downloads with Progress Bar

If you'd like to download larger files from the Internet and get a progress indicator, you can load the .NET Visual Basic assemblies, which provide a sophisticated download method with progress bar:

```
Add-Type -AssemblyName Microsoft.VisualBasic

$url = 'http://powershell.com/cs/media/p/26784/download.aspx'
$localPath = "$home\Documents\objects_and_types.pdf"

$object = New-Object Microsoft.VisualBasic.Devices.Network
$object.DownloadFile($url, $localPath, '', '', $true, 500, $true, 'DoNothing')


explorer.exe "/SELECT,$localPath"
Invoke-Item -Path $localPath
```

However, you may be wondering why the download only takes a fraction of a second, and why the downloaded file is very small. DownloadFile() does not support page redirects. The URL we used in this example is redirecting to the real download URL.

So this approach only works with the actual download URL, and fails if there are redirects involved.

4. Resolving Redirects

To find out the real URL and resolve redirects, use a Response object and start a low level communication. This will resolve the download URL we used in the previous examples:

```
$url = 'http://powershell.com/cs/media/p/26784/download.aspx'
$response = [System.Net.WebRequest]::Create($url).GetResponse()
$response.ResponseUri.OriginalString
$response.Close()
```

The result looks like this:

```
PS > $response.ResponseUri.OriginalString
$response.Close()

http://www.powershell.com/cs/PowerTips_Monthly_Volume_4.pdf
```

So the real URL looks quite different. You can now use this code to download redirected URLs with a progress bar:

```
Add-Type -AssemblyName Microsoft.VisualBasic

$url = 'http://powershell.com/cs/media/p/26784/download.aspx'
$localPath = "$home\Documents\objects_and_types.pdf"

$response = [System.Net.WebRequest]::Create($url).GetResponse()
$realurl = $response.ResponseUri.OriginalString
$response.Close()

$object = New-Object Microsoft.VisualBasic.Devices.Network
$object.DownloadFile($realurl, $localPath, '', '', $true, 500, $true, 'DoNothing')


explorer.exe "/SELECT,$localPath"
Invoke-Item -Path $localPath
```

5. Downloading with BitsTransfer

BITS is the technology used by Windows to download updates. It is designed to download even large files reliably, not fast. BITS can work across restarts. Eventually, over time, the file is downloaded.

You can use BITS to download files synchronously (while you wait) and asynchronously (in the background).

This will download yet another of our award-winning monthly guides, this time covering even more useful .NET types magic. It uses the synchronous method, and while the download takes place, you see a progress bar:

```
$url = 'http://powershell.com/cs/media/p/30542/download.aspx'
$target = "$HOME\Documents\PowerShell_Using_Registry.pdf"
```

Author Bio

Tobias Weltner is a long-term Microsoft PowerShell MVP, located in Germany. Weltner offers entry-level and advanced PowerShell classes throughout Europe, targeting mid- to large-sized enterprises. He just organized the first German PowerShell Community conference which was a great success and will be repeated next year (more on www.pscommunity.de).. His latest 950-page "PowerShell 3.0 Workshop" was recently released by Microsoft Press.

To find out more about public and in-house training, get in touch with him at tobias.weltner@email.de.

```
Import-Module BitsTransfer
Start-BitsTransfer -Source $url -Destination $target

explorer.exe "/SELECT,$target"
Invoke-Item -Path $target
```

6. Asynchronous Downloads with BITS

To reliably download very large files, you may want to use BITS in asynchronous mode. When you start a download this way, you do not have to wait for the download to complete. It will take place silently in the background, even across restarts.

It is your responsibility, though, to check back later and find out whether the download has completed. If it has, you must explicitly finalize the download. The download will not automatically be stored as a file.

This starts an asynchronous download and downloads yet another monthly guide, this time covering Windows Registry operations:

```
$url = 'http://powershell.com/cs/media/p/31297/download.aspx'
$target = "$HOME\Documents\PowerShell_Using_Registry.pdf"

Import-Module BitsTransfer
Start-BitsTransfer -Source $url -Destination $target -DisplayName MyDownload -Asynchronous
```

PowerShell will return control to you momentarily, and output something like this:

```
JobId                DisplayName   TransferType  JobState      OwnerAccount
-----                -----------   ------------  --------     ------------
4ef5a476-b475-4a4f... MyDownload    Download      Connecting   TobiasAir1\Tobias
```

To check the status of your download, try this anytime:

```
PS> Get-BitsTransfer -Name MyDownload

JobId                DisplayName   TransferType  JobState      OwnerAccount
-----                -----------   ------------  --------     ------------
4ef5a476-b475-4a4f... MyDownload    Download      Transferred  TobiasAir1\Tobias
```

Once the download has finished, finalize it like so:

```
Get-BitsTransfer -Name 'MyDownload' | Complete-BitsTransfer
```

Only now will the file be written to the target location you specified with the -Destination parameter.

7. Downloading with Invoke-WebRequest

PowerShell 3.0 and better comes with Invoke-WebRequest, a cmdlet that basically encapsulates a WebClient object and makes it even easier for you to download and configure details like proxy servers or submitting credentials.

This will download yet another of our popular PowerShell guides, this time about regular expressions:

```
$Source = 'http://powershell.com/cs/media/p/29098/download.aspx'
$localPath = "$home\Documents\PowerShell_regex.pdf"

Invoke-WebRequest -Uri $Source -OutFile $localPath

explorer.exe "/SELECT,$localPath"
Invoke-Item -Path $localPath
```

And this will download the SysInternals suite of tools to your computer:

```
$Source = 'http://download.sysinternals.com/files/SysinternalsSuite.zip'
$Destination = "$env:temp\sysinternalssuite.zip"
Invoke-WebRequest -Uri $Source -OutFile $Destination
Unblock-File $Destination
```

Since downloaded files are blocked by Windows, PowerShell 3.0 comes with yet another new cmdlet; Unblock-File removes the block. Now you're ready to unzip the file.

If your Internet connection requires proxy settings or authentication, take a look at the parameters supported by Invoke-WebRequest.

9. Unblocking Downloaded Files

Any file you download from the Internet or receive via email gets marked by Windows as potentially unsafe. If the file contains executables or binaries, they will not run until you unblock the file.

PowerShell 3.0 and better can identify files with a "download mark":

```
Get-ChildItem -Path $Home\Downloads -Recurse |
  Get-Item -Stream Zone.Identifier -ErrorAction Ignore |
  Select-Object -ExpandProperty FileName |
  Get-Item
```

You may not receive any files (if there are none with a download mark), or you may get tons of files (which can be an indication that you unpacked a downloaded ZIP file and forgot to unblock it before).

To remove the blocking, use the Unblock-File cmdlet. This would unblock all files in your Downloads folder that are currently blocked (not touching any other files):

```
Get-ChildItem -Path $Home\Downloads -Recurse |
  Get-Item -Stream Zone.Identifier -ErrorAction Ignore |
  Select-Object -ExpandProperty FileName |
  Get-Item |
  Unblock-File
```

Technical Editor Bio

Aleksandar Nikolic, Microsoft MVP for Windows PowerShell, a frequent speaker at the conferences (Microsoft Sinergija, PowerShell Deep Dive, NYC Techstravaganza, KulenDayz, PowerShell Summit) and the co-founder and editor of the PowerShell Magazine (http://powershellmagazine.com). He is also available for one-on-one online PowerShell trainings. You can find him on Twitter: https://twitter.com/alexandair

10. Download Web Page Content

Probably the easiest way of reading raw web page content is using the WebClient object. This will read the web content of http://blogs.msdn.com/b/powershell/, the PowerShell team blog:

```
$url = 'http://blogs.msdn.com/b/powershell/'
$wc = New-Object System.Net.WebClient
$wc.DownloadString($url)
```

11. Ripping All Links from a Website

Invoke-WebRequest does not only download data, it also analyzes the data for you and does basic parsing, thus opening a window with all links on that website is a piece of cake:

```
$website = Invoke-WebRequest -UseBasicParsing -Uri http://www.cnn.com
$website.Links | Out-GridView
```

12. Scraping Information from Blogs and Web Pages

Regular expressions are a great way of identifying and retrieving text patterns. Take a look at the next code fragment as it defines a RegEx engine that searches for HTML divs with a "post-summary" attribute, then reads the PowerShell team blog and returns all summaries from all posts in clear text:

```
$regex = [RegEx]'<div class="post-summary">(.*?)</div>'

$url = 'http://blogs.msdn.com/b/powershell/'
$wc = New-Object System.Net.WebClient
$content = $wc.DownloadString($url)

$regex.Matches($content) | Foreach-Object { $_.Groups[1].Value }
```

It really is just a matter of finding the right "anchors" to define the start and end of what you are seeking. The next code segment reads all PowerShell team blog headers:

```
$regex = [RegEx]'<span></span>(.*?)</a></h4>'

$url = 'http://blogs.msdn.com/b/powershell/'
$wc = New-Object System.Net.WebClient
$content = $wc.DownloadString($url)

$regex.Matches($content) | ForEach-Object { $_.Groups[1].Value }
```

```
Configuring a SQL High Availability Group with DSC
DSC Diagnostics Moduleâ€“ Analyze DSC Logs instantly now!
Need more DSC Resources?  Announcing DSC Resource Kit Wave 2
How to enable Updatable Help for your PowerShell Module
Want to secure credentials in Windows PowerShell Desired State Configuration?
Separating &amp;quot;What&amp;quot; from &amp;quot;Where&amp;quot; in PowerShell DSC
Using Event Logs to Diagnose Errors in Desired State Configuration
Holiday Gift â€“ Desired State Configuration (DSC) Resource Kit Wave-1
Automatically resuming Windows PowerShell Workflow jobs at logon
Windows PowerShell Script Workflow Debugging
Windows PowerShell Remote Debugging
PowerShell Security Best Practices
(...)
```

13. Getting RSS Feed Information

If your company is using an Internet proxy, and you'd like to access Internet with a WebClient object, make sure it uses the proxy and supplies your default credentials to it. You could write a little helper function to get such a pre-configured WebClient:

```
function Get-Webclient
{
    $wc = New-Object Net.WebClient
    $wc.UseDefaultCredentials = $true
    $wc.Proxy.Credentials = $wc.Credentials
    $wc
}
```

Next, you could retrieve an RSS feed from the Internet, and convert it to an XML object for easy parsing:

```
$webclient = Get-WebClient

[xml]$powershelltips =
$webClient.DownloadString('http://powershell.com/cs/blogs/tips/rss.aspx')

$powershelltips.rss.channel.item | Select-Object title, link
```

14. Getting Up-to-Date Exchange Rates

XML objects provide a method called Load() that can load both file-based and URL-based XML data. So if a web page returns XML data (such as RSS feeds or REST APIs), you can use an XML object to download and process the information. Note that the XML object does not expose ways to specify a proxy server or credentials.

If you would like to get currency exchange rates, you can simply access a Web service from one of the major banks. Here is how to get the USD exchange rate needed to convert to Euro:

```
$xml = New-Object xml
$xml.Load('http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml')
$rates = $xml.Envelope.Cube.Cube.Cube

"Current USD exchange rate:"

$usd = $rates | Where-Object { $_.currency -eq 'USD' } |
  Select-Object -ExpandProperty rate

$usd
```

And this creates a comprehensive list of all exchange rates by using a hash table:

```
$xml = New-Object xml
$xml.Load('http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml')
```

```
$allrates = @{}
$xml.Envelope.Cube.Cube.Cube | ForEach-Object {
$currency = $_.currency
$rate = $_.rate
$allrates.$currency = $rate
}
“All exchange rates:”
$allrates
“Specific exchange rate:”
$allrates[“IDR”]
```

The result may look similar to this:

```
All exchange rates:

Name                           Value
----                           -----
BRL                            3.2837
TRY                            2.9962
LTL                            3.4528
MYR                            4.5160
RUB                            48.0840
KRW                            1453.10
JPY                            139.48
INR                            84.9080
(,,,)
```

15. Finding Popular Historic First Names

To find popular first names for given decades, check out the function Get-PopularName. It accepts a decade between 1880 and 2000 and then uses the new and awesome Invoke-WebRequest in PowerShell 3.0 to visit a statistical website and retrieve popular first names using a regular expression.

Note: if your Internet connection requires a proxy server and/or authentication, please add the appropriate parameters to Invoke-WebRequest.

```
function Get-PopularName
{
  param
  (
   [ValidateSet(‘1880’,’1890’,’1900’,’1910’,’1920’,’1930’,’1940’,’1950’,’1960’,’1970’,
’1980’,’1990’,’2000’)]
    $Decade = ‘1950’
  )

  $regex = [regex]’(?si)<td>(\d{1,3})</td>\s*?<td align=”center”>(.*?)</td>\s*?<td>((?:\
d{0,3}\,)*\d{1,3})</td>\s*?<td align=”center”>(.*?)</td>\s*?<td>((?:\d{0,3}\,)*\d{1,3})</
td></tr>’

  $web = Invoke-WebRequest -UseBasicParsing -Uri
“http://www.ssa.gov/OACT/babynames/decades/names$($decade)s.html”

  $html = $web.Content
  $Matches = $regex.Matches($html)
  $matches | ForEach-Object {
    $rv = New-Object PSObject | Select-Object -Property Name, Rank, Number, Gender
    $rv.Rank = [int]$_.Groups[1].Value
    $rv.Gender = ‘m’
    $rv.Name = $_.Groups[2].Value
    $rv.Number = [int]$_.Groups[3].Value
    $rv
```

```
    $rv = New-Object PSObject | Select-Object -Property Name, Rank, Number, Gender
    $rv.Rank = [int]$_.Groups[1].Value
    $rv.Gender = 'f'
    $rv.Name = $_.Groups[4].Value
    $rv.Number = [int]$_.Groups[5].Value
    $rv
  } | Sort-Object Name, Rank
}
```

Just for the fun of it, call Get-PopularName and send it to Out-GridView:

```
Get-PopularName -Decade 1960 | Out-GridView
```

Or dump it to the console:

```
PS C:\Users\Tobias> Get-PopularName -Decade 1950

Name         Rank Number Gender
----         ---- ------ ------
Alan           45  83928 m
Albert         76  52804 m
Alfred        113  29714 m
Alice          85  52059 f
Allan         169  17684 m
Allen          84  47285 m
Alvin         139  23059 m
Amy           135  30589 f
(...)
```

And these are the five most popular US names in the '30s:

```
PS> Get-PopularName -Decade 1930 | Sort-Object -Property Rank -Descending | Select-Object
-First 5

Name    Rank Number Gender
----    ---- ------ ------
Alex     200   6304 m
Daisy    200   9058 f
Johnnie  199   9059 f
Jon      199   6362 m
Nelson   198   6472 m
```

16. Search and View PowerShell Videos

PowerShell is amazing. It will let you search YouTube videos for keywords you select, and then offer the videos to you, and upon selection play the videos as well.

Here's a little script that--Internet access assumed--lists the most recent "Learn PowerShell" videos from YouTube. The list opens in a grid view window, so you can use the full text search at the top or sort columns until you find the video you want to give a try.

Next, click the video to select it, and then click "OK" in the lower-right corner of the grid.

PowerShell will launch your web browser and play the video. Awesome!

```
$keyword = "Learn PowerShell"

Invoke-RestMethod -Uri
"https://gdata.youtube.com/feeds/api/videos?v=2&q=$($keyword.Replace(' ','+'))" |
Select-Object -Property Title, @{N='Author';E={$_.Author.Name}}, @{N='Link';E={$_.Content.
src}}, @{N='Updated';E={[DateTime]$_.Updated}} |
Sort-Object -Property Updated -Descending |
```

```
Out-GridView -Title "Select your '$Keyword' video, then click OK to view." -PassThru |
ForEach-Object { Start-Process $_.Link }
```

Simply change the variable $keyword in the first line to search for different videos or topics.
Note that due to a bug in PowerShell 3.0, Invoke-RestMethod will only return half of the results. In PowerShell 4.0, this bug was fixed.

17. Getting Most Recent Earthquakes

Everything is connected these days. PowerShell can retrieve public data from web services. So here's a one-liner that gets you a list of the most recently detected earthquakes and their magnitude:

```
Invoke-RestMethod -URI "http://www.seismi.org/api/eqs" |
  Select-Object -First 30 -ExpandProperty Earthquakes |
  Out-GridView
```

18. Getting Excuses Automatically

Tired of inventing lame excuses? Then here's a script that gets you a new excuse any time you call Get-Excuse! All you need is Internet access:

```
function Get-Excuse
{
  $url = 'http://pages.cs.wisc.edu/~ballard/bofh/bofhserver.pl'
  $ProgressPreference = 'SilentlyContinue'
  $page = Invoke-WebRequest -Uri $url -UseBasicParsing
  $pattern = '<br><font size = "\+2">(.+)'

  if ($page.Content -match $pattern)
  {
    $matches[1]
  }
}
```

If your Internet access goes through a proxy or needs authentication, then look at the parameters of Invoke-WebRequest inside the function. You can submit proxy information as well as credentials.

19. Validating a URL

To make sure user input is a valid URL, use the System.URI type. Try to convert the raw string into this type. If it works, the string is a valid URL. You can then further examine the converted result to limit validation to only HTTP/HTTPS URLs:

```
function isURI($address)
{($address -as [System.URI]).AbsoluteURI -ne $null}

function isURIWeb($address)
{$uri = $address -as [System.URI]
$uri.AbsoluteURI -ne $null -and $uri.Scheme -match '[http|https]'}
```

Here are a couple of example calls:

```
isURI('http://www.powershell.com')
isURI('test')
isURI($null)
isURI('zzz://zumsel.zum')
isURIWeb('http://www.powershell.com')
isURIWeb('test')
isURIWeb($null)
isURIWeb('zzz://zumsel.zum')
```

20. Analyzing URLs

URLs contain a lot of information which can be automatically parsed by PowerShell. Simply convert a URL to the System.URI type.

Once you did this, all information contained in the URL is accessible via individual properties:

```
$result = [System.uri]'http://powershell.com/cs/forums/'
$result
```

```
PS > $result

AbsolutePath   : /cs/forums/
AbsoluteUri    : http://powershell.com/cs/forums/
LocalPath      : /cs/forums/
Authority      : powershell.com
HostNameType   : Dns
IsDefaultPort  : True
IsFile         : False
IsLoopback     : False
PathAndQuery   : /cs/forums/
Segments       : {/, cs/, forums/}
IsUnc          : False
Host           : powershell.com
Port           : 80
Query          :
Fragment       :
Scheme         : http
OriginalString : http://powershell.com/cs/forums/
DnsSafeHost    : powershell.com
IsAbsoluteUri  : True
UserEscaped    : False
UserInfo
```

```
"You are using port $($result.port)."
"Page comes from $($result.Authority)."
"Served using $($result.Scheme)."
```

```
You are using Port 80.
Page comes from powershell.com.
Served using http.
```

21. URL Encoding and Decoding

To encode and decode URL information, use this code:

```
[System.Web.HttpUtility]::UrlEncode('ÄÖÜ')
[System.Web.HttpUtility]::UrlDecode('%c3%84%c3%96%c3%9c')
```

```
PS> [System.Web.HttpUtility]::UrlEncode('ÄÖÜ')
%c3%84%c3%96%c3%9c

PS> [System.Web.HttpUtility]::UrlDecode('%c3%84%c3%96%c3%9c')
ÄÖÜ
```

23. HTML Encoding and Decoding

To encode and decode special characters for HTML content, use these methods:

```
[System.Web.HttpUtility]::HTMLEncode('This is a test & a good way to encode. ÄÖÜ')
[System.Web.HttpUtility]::HTMLDecode('This is a test &amp; a good way to encode.
&#196;&#214;&#220;')
```

```
PS> [System.Web.HttpUtility]::HTMLEncode('This is a test & a good way to encode. ÄÖÜ')
This is a test &amp; a good way to encode. &#196;&#214;&#220;

PS> [System.Web.HttpUtility]::HTMLDecode('This is a test &amp; a good way to encode.
&#196;&#214;&#220;')
This is a test & a good way to encode. ÄÖÜ
```

24. Accessing Internet Explorer

Accessing Internet Explorer can be useful to easily get access to the DOM (Document Object Model) of a web page. The usual approach uses a COM object called InternetExplorer.Application like this:

```
$ie = New-Object -ComObject InternetExplorer.Application
$ie.Visible = $true
$ie.Navigate('http://www.powershell.com')

$ie
$ie.Document
```

Unfortunately, this approach fails when you use IE with User Account Control (UAC) and context switching.

While you can open IE and navigate to the web page, you will lose access to the document property once the website is loaded because the COM object works only for local files. Once you navigate to a web page, IE launches again with lower privileges, and PowerShell has no way of accessing the new instance.

To work around this issue, you can use yet another COM object called Shell.Application. It returns all Explorer windows, including all open IE windows (no other browser bands).

So you could even launch Internet Explorer, browse to a site manually, do whatever login you want manually, and then have PowerShell connect to it and automate the content.

All you need do is pick the IE window you need, by specifying a keyword found in its title bar.

The following example launches a web page and then waits until a browser window opened with the word "PowerShell" in its title bar. Next, the code returns the IE object and accesses the web page content.

```
& "$env:programfiles\Internet Explorer\iexplore.exe" 'http://powershell.com'

$shell = New-Object -ComObject Shell.Application

$null = Read-Host 'Press ENTER when content is loaded in IE'

$IE = $shell.windows() |
  Where-Object { $_.locationName -like '*PowerShell*' } |
  Select-Object -First 1

$IE.document
```

25. Testing Open Browser Windows

The Shell.Application COM object returns a list of all open windows, including all opened IE browser windows. This way, you can find out whether a certain URL is visible in any browser window or not.

Check-BrowserURL accepts any URL (or part of it) and returns the number of IE browser windows using this URL:

```
function Check-BrowserURL($url)
{
    $shell = New-Object -ComObject Shell.Application
    $result = $shell.windows() |
      Where-Object { $_.locationURL -like "*$url*" }
    $result -ne $null
}

Check-BrowserURL powershell.com
Check-BrowserURL google.com
```

26. Sending POST Data via PowerShell

Often, feedback or votes on web pages are sent back via POST requests. You can send this kind of information via PowerShell as well. You should simply create a POST request with the target URL and the desired parameters/information and send it off:

```
$url = "http://someurl"
$parameters = "vote=true&poll_id=2" # your POST parameters

$http_request = New-Object -ComObject Msxml2.XMLHTTP
$http_request.Open('POST', $url, $false)
$http_request.SetRequestHeader("Content-type", "application/x-www-form-urlencoded")
$http_request.SetRequestHeader("Content-length", $parameters.length)
$http_request.SetRequestHeader("Connection", "close")
$http_request.Send($parameters)
$http_request.StatusText
```

27. Refreshing Web Pages

Imagine you opened a number of web pages in Internet Explorer and would like to keep the display current. Instead of manually reloading these pages in intervals, you can use this script.

Note: it will refresh Internet Explorer browsers only, and it needs to run in a PowerShell console, not the ISE editor.

```
function Refresh-WebPages {
    param(
        $interval = 5
    )

    "Refreshing IE Windows every $interval seconds."
    "Press any key to stop."

    $shell = New-Object -ComObject Shell.Application
    do {
        'Refreshing ALL HTML'
        $shell.Windows() |
        Where-Object { $_.Document.url } |
        ForEach-Object { $_.Refresh() }
        Start-Sleep -Seconds $interval
    } until ( [System.Console]::KeyAvailable )

    [System.Console]::ReadKey($true) | Out-Null
}
```

It will automatically refresh all opened Internet Explorer pages every five seconds.

28. Testing URLs for Proxy Bypass

If you'd like to find out whether a given URL goes through a proxy or is accessed directly, you can use this neat little trick:

```
Function Test-ProxyBypass
{
  param
  (
    [Parameter(Mandatory=$true)]
    [string]$url
  )

  $webclient = New-Object System.Net.WebClient
  return $webclient.Proxy.IsBypassed($url)
}
```

To test a URL, simply submit it to the function Test-ProxyBypass. The result is either $true or $false.