Photo Credit: PowerShell Magazine

PowerShell REST API Programming

Reaching to web servies on the web gives scripts data power

Being able to get and send data within a PowerShell script enables them to be NOT static. Communication with APIs enable PowerShell scripts to:

- Get input data

- Send SMS messages and voicemails to any phone
- Send emails
- Send calendar appointments

There are now two major flavors of how requests and responses are formatted:

- REST API, such as PowerShellforGitHub
- GraphQL, such as PowerShellforGitHubGraphQL

PowerShell has two commands to make web services calls:

- A lower-level Invoke-RestMethod.
- A higher-level Invoke-WebMethod.

Invoke-RestMethod documentation

Documentation on Invoke-RestMethod says the cmdlet was introduced in PS 3.0 to send HTTP and HTTPS requests to Representational State Transfer (REST) web services that returns richly structured data. No short alias is specified for it.

Its general syntax:

```
Invoke-RestMethod
[-Uri] <Uri>
[-Headers <IDictionary> ]
[-Body <Object> ]
[-Certificate <X509Certificate> ] [-CertificateThumbprint <String> ]
[-ContentType <String> ]
[-Credential <PSCredential> ]
[-DisableKeepAlive]
[-InFile <String> ]
[-MaximumRedirection <Int32> ]
[-Method <WebRequestMethod> {Default | Get | Head | Post | Put | Delete | Trace | Options | Merge | Patch} ] [-OutF
[-PassThru]
[-Proxy <Uri> ] [-ProxyCredential <PSCredential> ] [-ProxyUseDefaultCredentials]
[-SessionVariable <String> ] [-TimeoutSec <Int32> ]
[-TransferEncoding <String> {chunked | compress | deflate | gzip | identity} ] [-UseBasicParsing] [-UseDefaultCrede
[-UserAgent <String> ]
[-WebSession <WebRequestSession> ]
[ <CommonParameters>]
```

HTTP requests have both a GET and POST approach.

GET Invoke-RestMethod

Web services which do not require some registration is getting more rare nowadays.

But in this blog, Jeff Hicks found that NewEgg still has an RSS feed for their daily deals at http://www.newegg.com/RSS/Index.aspx

1. Run a basic HTTP GET in a PowerShell script containing:

```
$uri = "http://www.newegg.com/Product/RSS.aspx?Submit=RSSDailyDeals&Depa=0"
$response = Invoke-RestMethod -Uri $uri
$response.title
$response.count
$response[0] | format-list
```

BTW: Although “uri” means Universal Resource Identifier, to be technically correct, the inclusion of the access mechanism “http” makes it really is a URL, a type of URI.

PROTIP: When output to a variable, show a count of how many items were returned into the response variable so that it doesn’t look like nothing happened.

The response is overwhelming, so `format-list` is used to filter out just the first item returned in the response

PROTIP: Lists starts from zero.

This enables us to see the names of properties: Published, title, link

```
103
Published : 11/3/16 4:05:46 AM
title     : $199.99 - SHIELD Series RSCM-0916B081 - 16-Channel 960H,
          H.264-Level DVR Surveillance Kit + Eight 900TVL Cameras - Night
          Vision Up to 65 Feet, Remote Viewing Supported (HDD Not Included)
link      : http://www.newegg.com/Product/Product.aspx?Item=N82E16881147042&nm_
          mc=OTC-RSS&cm_sp=OTC-RSS-_-Surveillance%20Security%20Systems-_-SHIE
          LDeye-_-N82E16881147042
```

2. BLAH: I get an error from this, but it's here for future referene.

   To make the response clickable, feed the response through the `out-gridview` cmdlet introduced with v2 with an alias of ogv:

```
$response | ogv -Title "Deal of the Day" -OutputMode Multiple | foreach { Start $_.link }


[regex]$rx = "(?.)(?\d+\.\d{2})\s-\s(?.*)"
</strong></pre>
```

3. Another HTTP GET example (that no longer works) is:

```
$url="http://www.seismi.org/api/eqs"
$response = Invoke-RestMethod -Uri $uri
# Extract child items:
$people = $response.items
```

   This returns:

```
count earthquakes
----- -----------
21740 {@{src=us; eqid=c000is61; timedate=2013-07-29 22:22:48; lat=7.6413; lo...
```

POST Invoke-RestMethod

POST involve sending both a body and headers.

```
$person = @{
   first='joe'
   lastname='doe'
}
$body = (ConvertTo-Json $person)
$hdrs = @{}
$hdrs.Add("X-API-KEY","???")
$hdrs.Add("X-SIGNATURE","234j123l4kl23j41l23k4j")
$hdrs.Add("X-DATE","12/29/2016")
Invoke-RestMethod -Uri $url -Method Post -Body $body -ContentType 'application/json' -Headers $hdrs
```

ConvertTo-Json

The above on several lines is easier to read than one long line:

```
$hdrs = @{"X-API-KEY"='???'; "X-SIGNATURE"='234j123l4kl23j41l23k4j'; X-DATE"='12/29/2016'"}
```

The power of Powershell vs wget are such helpers and how it can fluidly turn input into objects, and then to manipulate those objects in a granular way.

The API-KEY is obtained from the service's website during sign-up.

WARNING: If `-ContentType 'application/json` is not added to REST calls, an error message is likely because when POST is specified, Invoke-RestMethod sets the content type to “application/x-www-form-urlencoded” for sending out forms, not REST calls.

Base64 Encoding for Authentication

See The example at Profitbricks for an example. You won’t be able to run the code if you don’t have an account.

But you’ll want to get an account because it’s a great service that is convenient and enables you to work with multiple clouds.

Authentication

For session authentication with cookies, see https://community.qualys.com/docs/DOC-5594 based on https://www.qualys.com/docs/qualys-api-v2-user-guide.pdf

1. Define credentials in environment variables:

```
$username = 'me_user'
$password = 'me_password'
$target = "Daily Whatsis Roundup"
```

2. Obtain a session variable `sess`

```
$hdrs = @{"X-Requested-With"="powershell"}
$base = "https://qualysapi.qualys.com/api/2.0/fo"
$body = "action=login&username=$username&password=$password"
Invoke-RestMethod -Headers $hdrs -Uri "$base/session/" -Method Post -Body $body -SessionVariable sess
```

   This doesn’t work anymore https://community.qualys.com/docs/DOC-4523#jive_content_id_Windows_Powershell_30

```
$username = "username"
$password = "password"
$password_base64 = ConvertTo-SecureString $password -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential ($username, $password_base64)
$headers = @{"X-Requested-With"="powershell"}
$url = "https://qualysapi.qualys.com/about.php"
Invoke-RestMethod -Headers $headers -Uri $url -Method Post -Credential $creds -OutFile response.xml
```

Basic Authentication to GitHub

The code below makes a request sending the credentials in an Authorization header:

‘Basic [base64(“username:password”)]’

In PowerShell that would translate to something like:

```
function Get-BasicAuthCreds {
    param([string]$Username,[string]$Password)
    $AuthString = "{0}:{1}" -f $Username,$Password
    $AuthBytes  = [System.Text.Encoding]::Ascii.GetBytes($AuthString)
    return [Convert]::ToBase64String($AuthBytes)
}
$BasicCreds = Get-BasicAuthCreds -Username "Shaun" -Password "s3cr3t"
Invoke-WebRequest -Uri $GitHubUri -Headers @{"Authorization"="Basic $BasicCreds"}
```

Ignore Self-Signed Certs

http://www.datacore.com/RESTSupport-Webhelp/using_windows_powershell_as_a_rest_client.htm notes When using Windows PowerShell as a client, to avoid SSL Certificate trust issues if using HTTPS, enter this function in the PowerShell window:

```
function Ignore-SelfSignedCerts
{
    try
    {
        Write-Host "Adding TrustAllCertsPolicy type." -ForegroundColor White
        Add-Type -TypeDefinition  @"
        using System.Net;
        using System.Security.Cryptography.X509Certificates;
        public class TrustAllCertsPolicy : ICertificatePolicy
        {
             public bool CheckValidationResult(
             ServicePoint srvPoint, X509Certificate certificate,
             WebRequest request, int certificateProblem)
             {
                 return true;
            }
        }
"@
        Write-Host "TrustAllCertsPolicy type added." -ForegroundColor White
      }
    catch
    {
        Write-Host $_ -ForegroundColor "Yellow"
    }
    [System.Net.ServicePointManager]::CertificatePolicy = New-Object TrustAllCertsPolicy
}
Ignore-SelfSignedCerts
```

Invoke-WebRequest to get file

Here is an example of downloading a file from the internet into whatever path is specified in the environment variable $Temp.

```
echo "$Temp=${env:Temp}"
Invoke-WebRequest -Uri 'https://oneget.org/nuget-anycpu-2.8.3.6.exe' -OutFile "${env:Temp}\nuget.exe"
```

The Hey Scripting Guy article from 2013 by Doug Finke, author of Windows PowerShell for Developers, offered this example, which now returns a “401 (gone)” because it’s deprecated. Nevertheless, try the syntax on a working API:

1. Filter the response through the `PSCustomObject` cmdlet and format it:

```
Invoke-RestMethod -Uri “https://gdata.youtube.com/feeds/api/videos?
v=2&q=PowerShell” | foreach {[PSCustomObject]@{Title=$.Title;
Author=$.Author.name; Link=$_.content.src}} | Format-List
```

Resources

https://www.jokecamp.com/blog/invoke-restmethod-powershell-examples/

http://www.powershellmagazine.com/2014/12/24/using-azure-resource-management-rest-api-in-powershell/

- Lee Holmes, author of Windows PowerShell Cookbook, 3rd Edition.
- http://www.powershellatoms.com/basic/download-file-website-powershell/

https://www.youtube.com/watch?v=U3Ne_yX4tYo&index=1&list=PL5uoqS92stXioZw-u-ze_NtvSo0k0K0kq
PowerShell Excel Module - ImportExcel Playlist of 5 vidoes Oct 5, 2017 by Doug Finke