LOOSE CANDY

PICK AND MIX POWER TIPS FOR .NET DEVELOPERS AND IT PROFESSIONALS.

IRIS CLASSON

Table of Contents

About the Author

Iris Classon is a passionate software developer and sought-after speaker. She is known for her ability to explain complex concepts in a simplified, personal, and engaging manner. As a clinical dietitian and fitness fanatic, she knows everything about accelerated learning as proven by her rapid career advancement as a developer, author, and trainer. Iris is a Microsoft MVP, Pluralsight author, and named as one of the top female influencers in tech. Her life is very much about contributing to the developer community, as well as being an eternal student striving towards pragmatic perfection in programming.

Preface

Sweden is known for their pick and mix candy, a phenomenon I have not come across in many countries. Most shops here that sell candy of some sort have a wall with assorted candy, and hopefully my favorite — the chewy crocodiles.

As a dietitian, I can tell you that the human mind is wired for sweet stuff. Sweets elicit endorphins as well as minimize the sense of pain. Maybe that is why I like to take in information in bite-size chunks. All at once it would simply be too overwhelming. On the contrary, bite-size pieces are easier to relish, maybe even leaving me wanting more. When I started learning programming that was my approach, and it still is.

I started writing this book as I was recovering from a personal crisis. Even when I was at my lowest point, hidden features and exciting new tools would pull me back into the mesmerizing world of programming — even if just for a moment. I treasured those moments. In a way, you could see them as my 'chewy crocodiles'. The blog slowed down to a crawl during that time, little if anything was written, but I kept a notebook with these little treats. Once out on the other side, the treats became my Tip of the Day, the collection you are holding in your hand. Since the O'Reilley book (co-authored) I wanted to write another one, and try self-publishing as a way to learn about the publishing industry. This book is a careful first step to try out self-publishing, well aware that it is far from perfect and coherent. I have done my best to price it accordingly, and all I ask for in return is your feedback so I can make the most out of this learning experience. As

always, I will share what I have learned along the way. I have always found humans, you and the rest of the community, to be the best source of up to date information — and certainly the most encouraging way to learn.

The book is not a guide, language reference, tutorial, or discussion. It is a book primarily concerned with loosely grouped advice and tips that rarely make it in to guides, tools, or language references. I am aware that some advice might have short expiry dates. Consequently, I have planned for continuous maintenance in order to keep this book as contemporary and enjoyable as possible. The e-book format allows me to update the book continuously, but hard copies are going to be a little bit tricky. For that reason, I recommend the e-book versus a hard copy.

I have never been good at goodbyes, and therefore I will simply say thank you, enjoy the book, and let me know what you think.

Chapter 1: PowerShell

PowerShell has been around for a good while although it had a slow adoption in the beginning. Today few would consider task automation and configuration management on Windows without PowerShell. Usage varies from extensive modules and scripts to smaller macros and background processes. If you have a problem, there is probably a PowerShell script for that.

This chapter covers a few tips and tricks to showcase PowerShell's abilities, from simple modules I find useful, to more involved scenarios such as working with SQL upgrade scripts and sending emails. You will also find PowerShell script sprinkled throughout the book, for example in the Azure Management chapter where we create resource notifications.

1.1 Modify Hosts File with the PSHosts Module

I often find myself working with host entries. Among other things, it provides a crafty way to emulate tenants and do network testing, block, or redirect sites. Searching for the file always bothered me so a simple module such as the `PsHosts` was very welcome. With the import of a module, I can do that work without editing the file directly.

After installing the module with `Install-Module PsHosts` cmdlet proceed with adding new host entries with `Add-HostEntry`.

```
Install-Module PsHosts

Add-HostEntry -Name 'tenant.domain' -Address 127.0.0.1

#Name            Address           Enabled
#----            ------            -------
#tenant.domain   127.0.0.1         True
```

The `Get-HostEntry` cmdlet lists all the host entries. To select a specific entry, specify the name of the entry.

```
Get-HostEntry -Name 'demo'

#Name            Address           Enabled
#----            ------            -------
#demo            127.0.0.1         False
```

Other cmdlets

`Disable-HostEntry` and `Enable-HostEntry` sets the host entry status while `Remove-HostEntry` removes the specified host entry

Important: if install-module fails
PowerShell v5 is required, check your version with `$PSVersionTable.PSVersion`. The PowerShell 5 update is included in Windows 10 as well as Windows Server 16 and up, and is also downloadable as a part of the Windows Management Framework 5.0.

1.2 Use PowerShell to Execute JavaScript on a Page

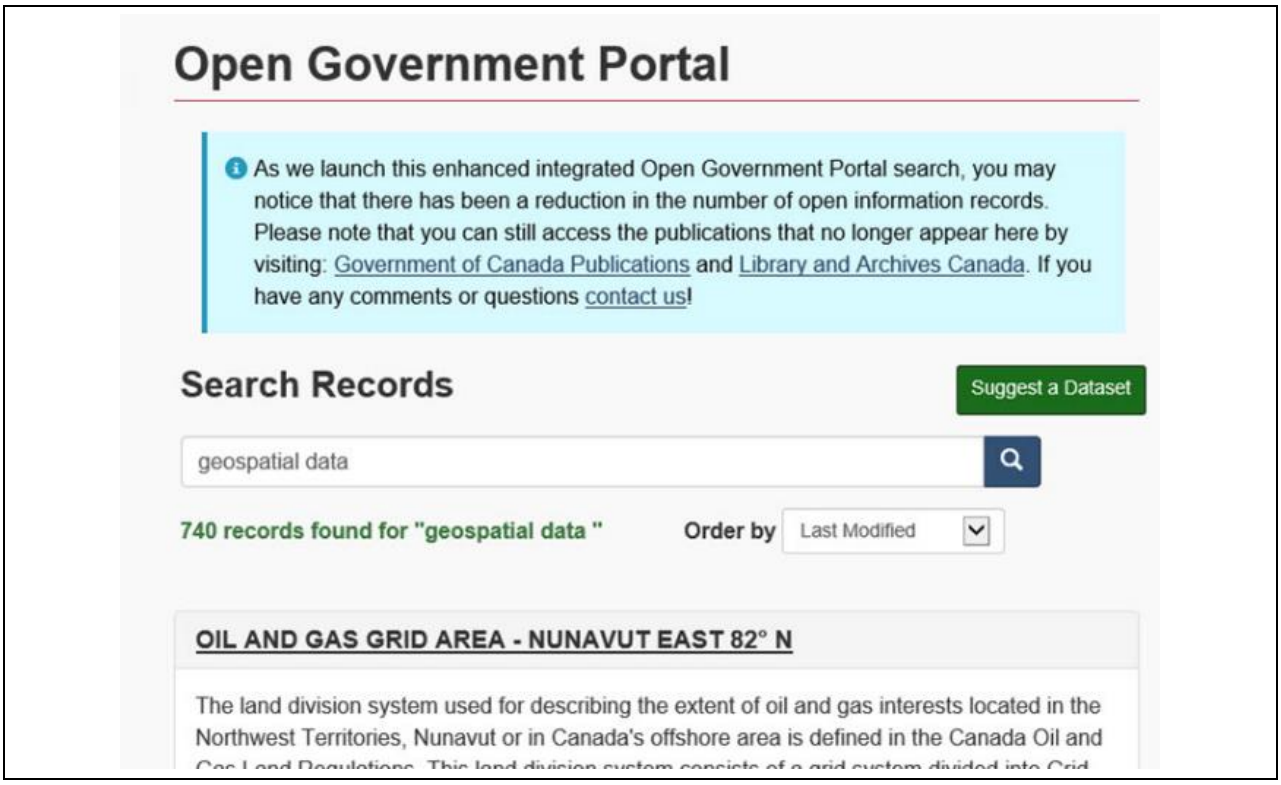
Although limited to IE, PowerShell lets you both scrape and execute JavaScript in the background with full access to the DOM and without Windows or tabs opening. If you want to watch the magic happen, you can set the visibility property as true.
Even better, you can execute JavaScript by injection- and afterward, grab the result. I have used this to generate screenshots as a part of end-to-end testing. Simply pass in the script as a string, and add *'JavaScript'* as the second parameter.

Let's walk through the example below.
I am creating a new IE object, navigating to the Canadian government open-data site, and doing a search. I wait for 5 seconds then grab the search results as titles and URLs.
At the end of the example, you can see that I am calling my screenshot script (script itself not included here) which sets the body of the page to the base64 encoded string.

Note: base64 encoded string
This is a binary-to-text encoding scheme that lets you represent binary data, in this case the image, in an ASCII string format

```
$url = 'http://open.canada.ca/data/en/dataset'

$ie = New-Object -ComObject InternetExplorer.Application

$ie.Navigate2($url)

$ie.Visible = $false

# Giving IE time to load

while ($ie.ReadyState -ne 4) {
    Start-Sleep -m 100 }

Start-Sleep -m 100

$document = $ie.document
$window = $document.parentWindow
```

```
$inputFields = $ie.document.body.getElementsByTagName('input')

$input =  $inputFields | Where-Object { $_.id -eq 'search_field'}

$input.value ='geospatial data'

$buttons = $ie.document.body.getElementsByTagName('button')

$button = $buttons | Where-Object { $_.className -match 'btn btn-primary btn-small'}

$button.click()

Start-Sleep -s 5

$allHeaders = $ie.document.body.getElementsByTagName('h3')

$searchItems = $allHeaders | Where-Object {$_.className -match "panel-title"}

$searchItems | % {
    $url = $_.getElementsByTagName('a')[0].href
    "TITLE: $($_.outerText) URL: $url"
}

$window.execScript($screenshotCode, 'javascript')

$window.execScript('screenshot()', 'javascript')


$window.document.body.innerHTML.ToString() | Out-File -Encoding default -
FilePath $file


# Output (10 links)
# TITLE: OIL AND GAS GRID AREA - NUNAVUT EAST 82° N  URL: http://open.canada.ca...
```



Screenshot of the website after the search

1.3 Add and Automate Startup Processes

At work, as our product grows, so does the complexity, and the number of things we need to know and that can go wrong. Automation is key to keep growing, and as much as possible we try to automate different tasks. One of them is the installment of MongoDB (a document oriented database program) as a startup process.

Below you will find a simplified example:

```
New-Item C:\Mongo\data\db
New-Item C:\Mongo\data\log

"" > C:\Mongo\data\log\mongod.log

@"
systemLog:
        destination: file
        path: C:\Mongo\data\log\mongod.log
storage:
        dbPath: C:\Mongo\data\db
"@ > "C:\Mongo\mongod.cfg"


# This is version 3.2
```

```
Invoke-WebRequest 'downloadUrl' -OutFile mongodb.msi

# /qn runs the install silently (with all defaults)
Start-Process mongodb.msi /qn -Wait

$binaryPathName = '"C:\Program Files\MongoDB\Server\3.2\bin\mongod.exe" --service --
config "C:\Mongo\mongod.cfg"'

New-Service -Name 'MongoDB' -BinaryPathName $binaryPathName -DisplayName 'MongoDB' -
StartupType Automatic

Start-Service -DisplayName 'MongoDB'
```

First attempt

I first gave Chocolatey* a try, but it would not install MongoDB even though the download would run to completion with no failures. I was able to solve this with PowerShell altogether.

> Note: Chocolatey
> A package manager for Windows that allows for easy and automated installs. Like apt-get, but for Windows.

MongoDB expects both DB and log directories to exist, and add an empty log file as well. The service might refuse to run giving you the following error:

```
PS C:\WINDOWS\system32> Start-Service -DisplayName 'MongoDB'
Start-
Service : Service 'MongoDB (MongoDB)' cannot be started due to the following error: C
annot start service MongoDB
on computer '.'.
At line:1 char:1
+ Start-Service -DisplayName "MongoDB"
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
+ CategoryInfo : OpenError: (System.ServiceProcess.ServiceController:ServiceControlle
r) [Start-Service],
ServiceCommandException
+ FullyQualifiedErrorId : CouldNotStartService,Microsoft.PowerShell.Commands.StartSer
viceCommand
```

If you encounter errors, you probably messed up a path, the markup in the config file, or put in a bad argument list for the start service registration. The latter error example might look familiar. Run the *'services.msc'* application and try starting the service from there to get an error number or a better description. Moreover, close the services.msc application before attempting to delete the faulted service as it is only marked for deletion but not deleted until you do so. Therefore, subsequent attempts at registering a service by the same name will fail.

1.4 Query and Update MS SQL Databases

One of my most popular blog posts is one on querying SQL server using PowerShell. It was written back in 2013, and a lot has changed since then, including invoking SQL queries with PowerShell. During a sprint, my development team had to update several tables across several databases in our development and quality assurance environment. What used to be a longer PowerShell script has become a denser script after the SQL cmdlets were introduced. Below you will find some simplified examples. For SQL calls to a local SQL server with integrated authentication credentials can be omitted and `Invoke-Sqlcmd` can be called directly with just the query text.

```
$databases | % {
     Invoke-Sqlcmd -Database $_ -ServerInstance $serverInstance -Username $username -
Password $pwd -Query $updateQuery
}

# Here is an example with variables:
$allVars = "ID='$tenantId'","DOMAIN='$Domain'"

$file = '.\Create_Empty_Schema.sql'

Invoke-Sqlcmd -InputFile $file -Variable $allVars -Database $Domain -
ServerInstance $serverInstance -username $userName -password $pwd
```

```
The variables are then used in the SQL script like this:

INSERT [dbo].[SystemSetting] ( [Setting], [Value]) VALUES ( N'TenantId', $(ID))
GO
INSERT [dbo].[SystemSetting] ( [Setting], [Value]) VALUES ( N'TenantDomain', $(DOMAIN
))
GO
```

Important: Quirks

There is an annoying quirk you should be aware of when passing in arguments to a script file. The arguments cannot contain an equal ("=") sign as the library used parses the argument key-value pairs on that char. The error will state that the argument is not formatted right- although it looks like it is and subsequently might drive you crazy.

1.5 Compare Database Schemas with SQL Server Tools

One of the many tasks I have automated on our build server is a script that compares database schemas and creates a report with the differences and if there is a risk of data loss. We have many different environments, release tracks and custom implementations for some clients. As a result, we have ended up with different database schemas, and it is easy to miss an update (currently done manually). Our email bot sends the report nightly and posts on our chat channel (Slack). The email contains the update script as an attachment and other information.
The automagical process that compares the databases and generates both update scripts and reports is *sqlpackage.exe*.

This very neat command line utility has many features, and the documentation covers them well. Through Visual Studio you can download the tool by adding SQL Server Tools to your installation or download it separately from MSDN. The toolset installs a scaled version of Visual Studio that is free of charge. The previous link explains that in greater detail.

To compare two databases, we need to extract two DACPAC (Data-tier Application Component Packages) packages using the databases as the source. DACPAC is just a way to package database changes. If you rename the extension to .zip and extract the files, you can see the XML files of which the package consists.

The executable is easier to call if aliased:

```
$path = 'C:\Program Files (x86)\Microsoft Visual Studio 14.0\Common7\IDE\Extensions\M
icrosoft\SQLDB\DAC\130\sqlpackage.exe'

Set-Alias -Name sqlpackage -Value $path
```

Afterward, I extract the packages by setting the source connection string, 'scs', and 'tf', which is short for 'to file'.

```
$qa = 'data source=...'
$prod = 'data source=...:'
$new = "$basePath\new.dacpac"
$old="$basePath\old.dacpac"

sqlpackage /a:Extract /scs:$qa /tf:$new
sqlpackage /a:Extract /scs:$prod /tf:$old
```

Once I have the packages, I can generate a deployment report, with some custom settings as I want to ignore and exclude a few things.

```
sqlpackage /a:DeployReport /sf:$new /tf:$old /tdn:"Konstrukt" /op:$xml /p:IgnoreRoleM
embership=true /p:ExcludeObjectTypes="Logins;" /p:IgnoreKeywordCasing=true

# tdn: target databasename
# op: output file for report (.xml)
```

The report consists of two parts, alerts and operations. Alerts are, for example data issues, and operations are actions such as alter, add, and drop.

To generate an upgrade script, I only need to change a few things:

```
sqlpackage /a:Script /sf:$new /tf:$old /tdn:"Konstrukt " /op:$sqlFile /p:DropObjectsN
otInSource=true /p:CompareUsingTargetCollation=true
```