PowerShell HTML Health Check: Demo

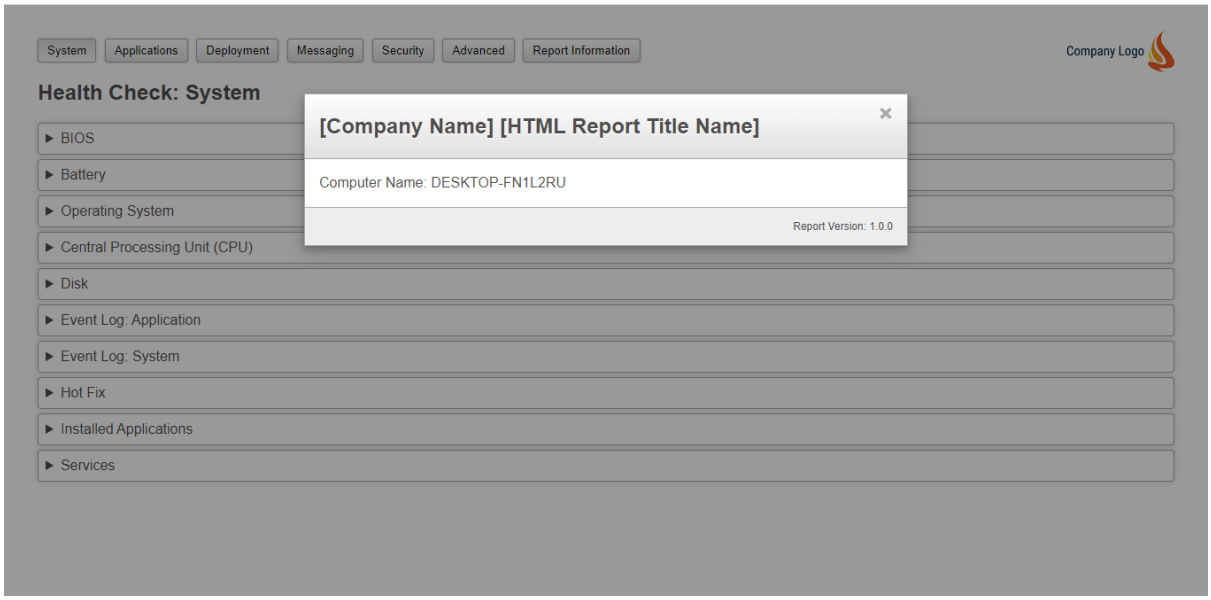
Project Name:	PowerShell HTML Health Check
Template Version:	1.0.0
Template Release Date:	28th May 2021
Author:	HerbsRy-Cyber (GitHub)
Repository:	https://github.com/HerbsRy-Cyber/PowerShellHTMLHealthCheck
Project Contact:	HerbsRyGitHub@GMail.com
Author Website:	www.HerbsRy.com
Tested OS's	Windows 10 1809, 1903, 1909, 20H2.
Release Notes:	Initial Release (V1)

In this demonstration I'll show you how to use the project template file to create a PowerShell script for generating a rich HTML report containing only the system information that's required by your organisation. The template does come pre-populated with some custom variables and some basic WMI queries / PowerShell cmdlets to get you started; here we'll explain how to use them, what sections of the script we need to maintain and how to use the functions that are at your disposal.

Contents

The Report Output Overview

The below image is the result of the report file being generated and launched:



When the report is first launched, a Modal dialog will display above the report displaying [Company Name], [HTML Report Title Name], Computer Name and the Report Version number. The [Company Name] data is pulled from one of the Custom variables available to you. The [HTML Report Title Name is also pulled from a Custom variable. The Computer Name will display by default as this shows you which computer the report was run on, however, the data displayed inside the body of the Modal can be manipulated as you please. Finally, the Report Version *Number* is also pulled from one of your *Custom* variables.

The buttons at the top of the report represent our main HTML navigation element, the *System* button is in a *clicked / pressed* state by default; when this button is clicked the area below the buttons will display the associated *Tab* area. Inside this *System Tab* area, we can display any information that we desire related to our System. As you can see, we have decided to display such information as BIOS, Battery, Operating System, CPU and so on. Each button that we click within the navigation area will display its own associated data to help you keep your report information separated and more organised.

In the top right corner of the report, you will see a *Company Logo*. You can choose to keep this area blank, or, as in the above image you can customise the report to display an image. The *Company Logo* is embedded in to the HTML page using a Base64 string rather than pulling from a file system location, this means that the image will always display and will not have to rely on your own logo being present on the system that has launched the report – this makes for a more portable script.

Each *Category*, such as BIOS, Battery etc. is automatically enclosed within a HTML <details> tag, meaning that when you click on a *Category* it will dynamically expand and reveal your queried information.

Required Custom Variables

At the top of the script file, you'll see the following area, this is the area in which you will add values to the existing variables to start customising your script.

```
# ------------------------------------------------------------------------
# Required :: Variables (Customisation Required)
# ------------------------------------------------------------------------

    $Author                  = "[Author Name]"
    $ReportVersion           = "1.0.0"
    $ConsoleTitle            = "[PowerShell Console Window Title]"
    $ReportTitle             = "[HTML Report Title Name]"
    $CompanyName             = "[Company Name]"
    $ReportLocation          = "$env:USERPROFILE\Desktop\"
    $ErrorActionPreference   = "SilentlyContinue"
    $CompanyLogoBorder       = "none"
    $CompanyLogoWidth        = "127px"
    $CompanyLogoHeight       = "50px"
    $CompanyLogoTop          = "28px"
    $CompanyLogoRight        = "37px"
    $CompanyLogoBase64String = "iVBORw0KGgoAAAANSUhEUgAAAH8AAAAy.."
```

- **Author**
  - This allows the author of the script to stamp their name in the HTML source code
- **Report Version**
  - This string displays on the Report Information *Modal*
- **Console Title**
  - This updates the Title string on the PowerShell console window
- **Company Name**
  - This string may display in various areas of the report such as the *Modal*
- **Report Location**
  - This instructs PowerShell where to create the report file and where to launch it from
- **ErrorActionPreference**
  - This should be self-explanatory to PowerShell users; using a value of *SilentlyContinue* will prevent the script from grinding to a halt in the event of an error and will allow the report to continue, thus, allowing the error to be caught in a variable that then populates a debugging section on the *Advanced Tab*.
- **Company Logo Border**
  - Currently set to 'none' to remove any border from the perimeter of your custom logo. This is a CSS value, therefore, when setting up your logo you can use such values as '1px dashed black' to display a thin black border around your logo to help you position it correctly before reverting the value back to 'none'
- **Company Logo [Width height Top Right]**
  - These values are CSS values; the position of the logo is set to a CSS value of 'Absolute', therefore the values set within these variables help configure both [Width] and

[Height] as well as position the logo in pixels from both [Top] and [Right] screen borders.

- **Company Logo Base64 String**
  - This Variable contains the value of your company logo.
  - If you do not wish to display an image, simply set the value to an empty string

Required :: Variables, Calculations, Expressions & Initial Configuration

In this area you'll see some basic configuration commands.

First, PowerShell will change the title of the console window to the value set in your Custom area, followed by displaying a progress bar (at 0%), followed by the creation of two new variables which allow you to use them as short(er)-hand within your script if you so desire. For example, you can now use `$ThisPC` instead of `$env:COMPUTERNAME`.

```
# ------------------------------------------------------------------------------
# Required :: Variables, Calculations, Expressions & Initial Configuration
# ------------------------------------------------------------------------------

Try { $Host.UI.RawUI.WindowTitle = $ConsoleTitle }
Catch {}
Finally { $Error.Clear(); Clear-Host; }

Write-Progress -Activity 'Generating Report'
               -Status   'Loading Script Variables'
               -PercentComplete 0;

Sleep -Milliseconds 750

$ThisPC      = $env:COMPUTERNAME

$CurrentUser = $env:USERNAME
```

Custom :: Variables, Calculations, Expressions & Initial Configuration

In this area you'll see that the default template holds some variables, the existing variables are used only to supply values to the demo queries further down the script. This is your area. Place all of your custom variables and expressions etc. here in one place so that you call upon them from any query below these variables. If you keep all of your custom variables here in one central place, it'll be easier for you to find them later.

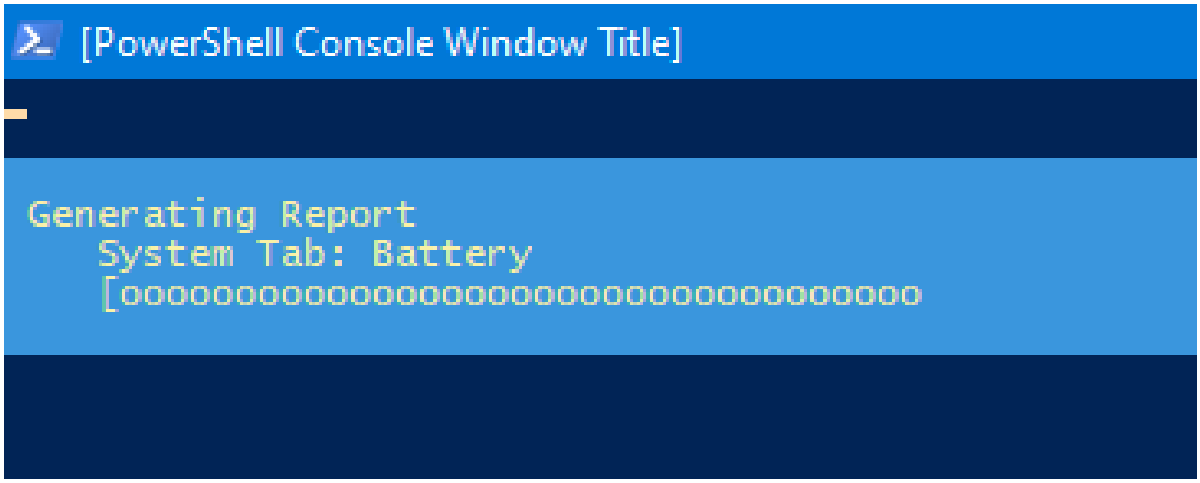
```
# --------------------------------------------------------------------------------
# Custom :: Variables, Calculations, Expressions & Initial Configuration
# --------------------------------------------------------------------------------

# This custom area allows you to keep your own variables in one place (Globally)
allowing them to be used from various locations below.
# The current variables in this section are here for the purpose of demonstrating
and are used by the existing template queries below.
```

Functions: Update-ProgressBar

This function allows you to update the Progress Bar within the PowerShell Console. It can be used at any time to update the progress bar text and value, this will help the agent running this script understand at what step the script is currently running its commands. Ideally, you should update the Progress Bar to specify what the script is doing right before you specify the command. In the image below we have updated the progress bar to state we are now querying the system Battery information and set the progress bar value to 20% - directly after the call to this function we query the Battery using WMI.

Example Usage:
Update-ProgressBar -Activity "Generating Report" -Status "System Tab: Battery" -PercentComplete 20;



Functions: Add-HTMLTableAttribute

This function allows you to add a custom attribute such as ID or Class to your auto-generated table(s). It will however auto-generate <col> and <colgroup> tags, therefore there is a script block within the $Scripts variable section that will iterate through the DOM and remove all instances of <col> and <colgroup> when the DOM is loading. The original Source will contain the elements but the Developer Tools (Live DOM) will remove them

Example Usage:

$WMI = [WMI Query] | ConvertTo-HTML -Fragment | Out-String | Add-HTMLTableAttribute -AttributeName 'id' -Value 'tab-system-cpu';

In the above example we have added an ID meta tag to the table with a value of ‘tab-system-cpu’. This allows us to then make use of any CSS styles that we have created, or, allows us to make use of any JavaScript commands to help manipulate and work with this specific ID. The results of the DOM would be as follows: <table id="tab-system-cpu">. . .</table>

Functions: Add-PreContentMessage

This function allows you to add Pre-Content HTML directly above the Table output by displaying a banner with text and an image. It allows you to change the Image, Colour and Text of the banner using one of the 3 switches (Below)

- Error
  - Red Banner, Red Text, traditional Error Icon
- Information
  - Blue Banner, Blue Text, traditional Information Icon
- Warning
  - Amber Banner, Amber Text, traditional Warning Icon

Example Usage:

$WMI = [WMI Query] | ConvertTo-HTML -Fragment -PreContent (Add-PreContentMessage -Type "Information" -Message "Displaying data for drive C: only.");

Example: Information Content on the Disk Category

▼ Disk

Displaying data for drive C: only.

DeviceID	VolumeName	Size (GB)	Free Space (GB)	Free (%)
C:		118	70	59

Example: Warning Content on the Disk Category

▼ Disk

Displaying data for drive C: only.

DeviceID	VolumeName	Size (GB)	Free Space (GB)	Free (%)
C:		118	70	59

Example: Error Content on the Disk Category

▼ Disk

Displaying data for drive C: only.

DeviceID	VolumeName	Size (GB)	Free Space (GB)	Free (%)
C:		118	70	59

Your Custom Queries

This is the area in which you will create your own commands to query the system and cram that data in to a variable that can be used at a later stage to output in to a Table.

Here, we'll take a look at the first query in the template file:

```
Update-ProgressBar -Activity "Generating Report"
                   -Status "System Tab: BIOS"
                   -PercentComplete 20;

$Query_System_BIOS = Get-WmiObject
                       -Class Win32_BIOS
                       -Computername $ThisPC |
                       Select SMBIOSBIOSVersion,
                              Manufacturer,
                              Name,
                              SerialNumber,
                              Version |
                       ConvertTo-HTML -Fragment;
```

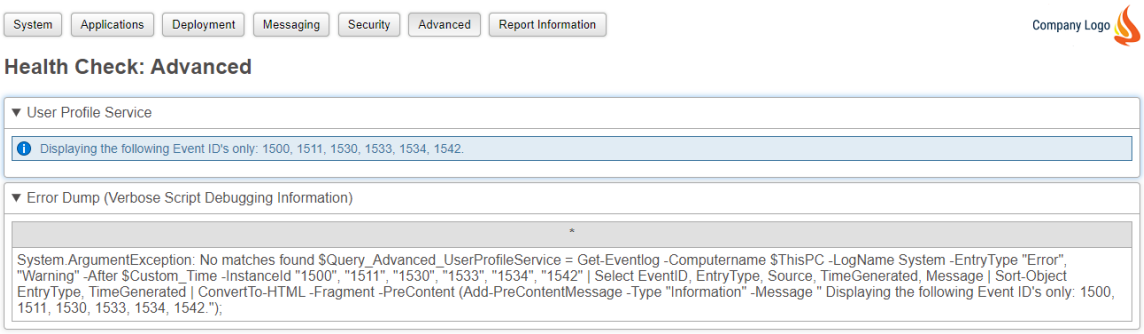
In this section, before we create a variable to query the BIOS data, we make a call to the Update-ProgressBar function so that we can update the PowerShell console and tell our user agent what's currently happening, directly followed by our variable named '$Query_System_BIOS' which consists of a standard Get-WMI command piped in to a cmdlet that converts the returned data in to a HTML Fragment (which will be a HTML Table). At this point, the query grabs the data and stuffs it in to a HTML string inside the variable – we will extract this data later when we look at the HTML script section.

The key takeaway here – for each *Category* that you want to display data for (example: BIOS), first update the progress bar with a friendly message, secondly, create a variable and set its command, third, pipe that command to a HTML Fragment, fourth (optional): either add a PreContent Message of create an ID / Class for the table data.

The Advanced Navigational Button

I like to use the Advanced area to display information for *Specific* events, rather than dumping a bunch of generic data for, let's say the BIOS. We could run another BIOS query here for the Advanced tab but only look for specific data that may be helpful to previous known issues, or well-known issues.

For example, there is by default a Category called 'User Profile Service' which will only display events from your system EventViewer based on a small set of Event ID's. These particular events are known to be related to User profile issues, therefore, if there is data present here, it's 100% worth looking in to as there could be some fundamental issues with *said* user profile. On the other hand, if there is no data present here, it instantly shows us that the current user profile is healthy and we can simply move on to the next category.



You'll also notice that by default there is a *Category* labelled 'Error Dump', this will contain all errors that were generated from your custom queries and help you understand issues with your script. The data present in the above image is an error relating to the User profile Service query. It's not so much an error with the script, rather, it's the error returned by the query to say there are no such events found in the event viewer – it does not, however, mean that there's a problem with your script. Each error caught while the script is running will be displayed in this category within its own row of the table to help separate and organise the output.

Required :: HTML :: DOM Elements :: Head Tag :: Includes Title, Styles

In this section of the script we lay out the <head> tags of our HTML report, this consists of all *CSS Styles*, the <title> tags, and a small comment block, for example: (The comment block can only be accessed by viewing the source of the HTML).

```
<!--
   Author   : $Author
   Date     : $Custom_CurrentDate
   Computer: $ThisPC
   Company  : $CompanyName
   Version  : $ReportVersion
-->
```

All of the CSS styles have been set for you so unless you want to change the colour of something there's nothing you need to do.

Required :: HTML :: DOM Elements :: Company Logo, Buttons & Tabs

In this section, the <body> of the HTML is structured, this is where the *Custom Queries / Variables* that you have created will be added, the placement of your variables determines on what navigational Tab your data appears. Let's take a look at what this section consists of:

$Modal

Notice in the <h2> tag it specifies the $CompanyName and $ReportTitle string values in the *Header*.

In the *Body* of the Modal, you'll see here inside a <p> tag it holds the computer name pulled from the $ThisPC variable. This is a standard *Paragraph* tag, using html you can manipulate the body of the Modal very easily.

The Modal *Footer* contains the $ReportVersion text.

All of the existing classes and related CSS values are both required and extracted from the CSS style section within the <head> of the report

```
$Modal = "
<aside id='modal' class='modal'>
    <div class='modal-content'>
        <div class='modal-header'>
            <span class='modal-close'>&times;</span>
            <h2>$CompanyName $ReportTitle</h2>
        </div>
        <div class='modal-body'>
            <p>Computer Name: $ThisPC</p>
        </div>
        <div class='modal-footer'>
            <p>Report Version: $ReportVersion</p>
        </div>
    </div>
</aside>
"
```

$CompanyLogo

Here we use a semantic <aside> tag to stamp an area of the DOM for our Company logo. This variable does not need to be modified

```
$CompanyLogo = "
<aside id='company-logo'></aside>
"
```

$ButtonElements

Here are the *Buttons* in our *Navigation* area, JavaScript is connected to each button so that when clicked the correct Tab area is hidden / displayed.

```
$ButtonElements = "
<nav>
<button class='button-clicked' onclick=opentab('tab-
system');navBtnClick(this);>System</button>
<button onclick=opentab('tab-applications');navBtnClick(this);>Applications</button>
<button onclick=opentab('tab-deployment');navBtnClick(this);>Deployment</button>
<button onclick=opentab('tab-messaging');navBtnClick(this);>Messaging</button>
<button onclick=opentab('tab-security');navBtnClick(this);>Security</button>
<button onclick=opentab('tab-advanced');navBtnClick(this);>Advanced</button>
<button onclick=openModal()>Report Information</button>
</nav>
"
```

$TabSystem

Here lies the skeleton HTML for each *Category.* Each *Category* will require a <details> tag, inside the details tag there will need to be a <summary> tag followed by the variable name of your query.

That's all that's required to turn your previous WMI queries in to a table that will automatically be placed inside a dynamically expanding details Category.

```
$TabSystem = "
    <section id='tab-system' class='tab'>
        <h2>Health Check: System</h2>
        <details>
            <summary>BIOS</summary>
            $Query_System_BIOS
        </details>
        <details>
            <summary>Battery</summary>
            $Query_System_Battery
        </details>
        <details>
            <summary>Operating System</summary>
            $Query_System_OS
        </details>
        <details>
            <summary>Central Processing Unit (CPU)</summary>
            $Query_System_CPU
        </details>
        <details>
            <summary>Disk</summary>
            $Query_System_Disk
        </details>
        <details>
            <summary>Event Log: Application</summary>
            $Query_System_AppEvent
        </details>
        <details>
            <summary>Event Log: System</summary>
            $Query_System_SysEvent
        </details>
        <details>
            <summary>Hot Fix</summary>
            $Query_System_Hotfix
        </details>
        <details>
            <summary>Installed Applications</summary>
            $Query_System_InstalledApps
        </details>
        <details>
            <summary>Services</summary>
            $Query_System_Services
        </details>
    </section>
```

Required :: HTML :: DOM Elements :: Script Tag

This section of the script contains all and any JavaScript code required for the HTML page to function.

It contains some *Required* code as well as some *Option* and *Custom* code.

Feel free to manipulate this area as you see fit.