QA Checks
Developers Guide

This document is intended for developers who want to extend the functionality of the QA Scripts.

It is assumed that you have a good working knowledge of PowerShell and are familiar with the existing QA checks. A good understanding of various tools and utilities of getting information out of the Windows operating system via WMI, the registry or other means as required.

Please make sure you read and understand the User Guide too. Some of the information there is duplicated in this document.

This document refers to version 3.17 and above of the scripts.

Contents

Overview

The QA checks came about as a need to verify the build of new servers for our various customers and their environments. All new server builds are done with a custom gold image; however this image still lacks many of the additional tools and configuration settings needed before it can be accepted into support.

Most of this extra tooling and configuration is automated, however checks are still needed to make sure each customer has their specific settings in place.

Technical Details

The scripts are written in the Microsoft PowerShell language, with version 2.0 in mind. This is the basic version installed by default on Windows Server 2008 R2.

The script will run on almost all Windows Operating systems, as long as PowerShell version 2.0 or greater is installed, and the PowerShell window is run with administrative privileges.

Supported Operating Systems

Windows Server 2008 R2
Windows Server 2012
Windows Server 2012 R2
Windows Server 2016

Unsupported Operating Systems

Windows 2003 Server
Windows Server 2008
Any non-server operating system

While the scripts will work and produce results on the above list, they are not supported and some scripts may fail.

The Checks

There are over 100 different checks split over 9 separate sections. These are executed whenever the QA script is executed against a server and can usually take anywhere between 30 seconds and a couple of minutes to complete. If you are checking multiple servers then this time is per server. The script is set to run up to 5 checks concurrently.

Each check is written to be as efficient as possible; however due to the nature of some of them they may take a little longer than normal. With this in mind, each individual check has a timeout of 60 seconds. This should give them plenty of time to complete their task.

For a full list of checks and sections, check the User Guide, they are listed in Appendix C.

Layout Overview

Folder Layout

The QA scripts are laid out in the following way:

```
\-- Server-QA-Checks
    \-- checks
            \-- accounts
            \-- compliance
            \-- …
    \-- documentation
    \-- engine
    \-- i18n
    \-- settings
```

Root Folder

The main folder contains the compiled QA scripts as well as the compiler and GUI Configurator. Do not make any changes to the compiled scripts unless you are doing quick testing in order to fix an issue.

Checks

The `checks` folder has several subfolders, one for each section. Each of these subfolders contains the actual scripts that you will be creating. When you create a script, make sure it's named correctly and in the right section. The names are:

`-acc-`	Accounts
`-com-`	Compliance
`-drv-`	Drives
`-hvh-`	Hyper-V Host
`-net-`	Network
`-reg-`	Regional
`-sec-`	Security
`-sys-`	System
`-vmw-`	Virtual Machines

Documentation

Any user or developer documentation lives here

Engine

This folder holds all the separate function files required by the checks as well as the main script engine. The functions are separated out in order increase code-reuse and stop any scripts from failing by having a particular function be different from the others. For a full list, see Appendix B. The main engine file does all the work of making sure the checks execute correctly and collates the results. Do not change these files without good reason as all changes will be overwritten when a new copy is downloaded from GIT.

i18n

This folder is used for any language specific strings that are used throughout the script. Currently only English UK is here, however future languages may appear - can you help translate.?

Settings

This holds the `default-settings.ini` file as well as any environment specific settings for your organisation. There can be any number of settings file located here.

Creating New Checks

Before creating a new check, first make sure what you want to do isn't already implemented. If you are duplicating a check, I may be able to help by modifying the existing one to help suit your needs. This is may not be possible in all cases, but it's worth asking.!

Important

Since the lowest supported operating system is Windows Server 2008 R2 which ships by default with PowerShell v2.0; all scripts must be written for this level of PowerShell. Newer PowerShell commands can be used, but only if there is a fall back to a version 2.0 command or workaround.

An example of this is c-sys-05-system-event-log where a PowerShell v4 method of collecting event logs can be used as it's much quicker, however a PowerShell v2 method is also listed.

Script File Names

When creating a new check, the file name is important. Files should be named according to the following guide:

```
c-xxx-yy-zzzz.ps1
```

Where:

`xxx`	The category label for the check,
`yy`	A unique zero-padded number that is next in sequence for the category,
`zzzz`	A short name for the check, all lowercase and spaces replaced with hyphens.

Example:

```
c-acc-01-local-users.ps1
```

The compiler will automatically pick up any new scripts that are created.

General Standards

When writing a new QA check, there are several guidelines that should be followed. While some of these guides are not set in stone, they will help to keep all checks to a certain standard level.

1. Each and every check must be tested on a clean Windows 2008 R2 server. Additional checks on other Windows operating systems must produce the same pass/fail results (where applicable),
2. Everything must be wrapped in a Try/Catch block. The Catch can either be ignored or captured correctly as a script error, or part of the check as needed.
3. There should be zero output from the check. All output must be suppressed and only a fully formed `$results` value should be returned.
4. Each check should perform one function only. An example of this is the two event log checks. One for the Application log and one for the System log.
5. The script must always return a `$results` object regardless of pass, fail or a Try/Catch issue.

Layout Standards

Each and every check tries to follow the same standard layout; this helps with debugging and organisation. The standard layout helps with reading each check to quickly see what it does and how it does it.

Comment Header

Every check has the same comment header block at the very top. This must be included in your check in order for the compiler to pick up the required information for the various parts of the script.

Open any of the existing checks and read through some of the headers. You should notice a pattern in the layout and formatting. Try to keep this going forward in your checks.

Description

This gives the full description of the check, explaining its purpose and any other information that might help the end user running the script.

Required-Inputs

This lists any inputs that the check is looking for, along with the descriptions for each input. Validation rules can also be added for when the GUI configurator is being used. (See Appendix C for details)

Default-Values

The default 'generic' values of all the check inputs specified above

Default-State

If your check should be enabled by default, set this value to "Enabled". Any other value will mark it as "Skip". This is used when the settings INI files have no record of this check.

Input-Description

This allows checkboxes and/or drop-down lists to have short description for each value. For example: an input value may just be '0' or '1'. This might not mean much, so descriptions can be added. See Appendix D for example screen shots.

Results

A list of the messages returned from you check for each of the result states (pass, warning, fail, etc.). These are used in the HTML hover help display.

Applies

The type of servers this check applies to. The current values are listed below, but there is no set standard. Used in the HTML hover help display and the GUI tool

- All Servers
- Virtual Servers
- Physical Serves
- Hyper-V Host Servers
- Domain Controllers

Required-Functions

A list of any required functions that your script relies on. Available functions are listed in Appendix B. If your check requires a function not listed, simply add it to the bottom of your script and set this to "None"

Existing Checks And Their Methods

The table below may help with creating new checks by reusing code from the existing ones. Some checks may be listed two or three times as they could use a multitude of methods.

Script Code	Method Used
	Single registry key
(todo)	Multiple registry keys
	Simple WMI query
	WMI query, looking for single entries
	WMI query looking for multiple entries
	WMI query, excluding multiple entries
	Execute external EXE application
	Execute external COM object
	Using custom required function
	Manual only check

Testing Your Script

Once you have created your script and want to test its output, you can do this in one of two ways.

1. Run the script as a separate standalone check,
2. Compile the entire QA checks into one QA script.

Option 1

This is the quicker option to perform but does require a small change to your check that must be removed before using as a final version.

Add the following code to the very top of your script:

```
1.  Function newResult { Return ( New-Object -TypeName PSObject -Property @{'server'='';
    'name'=''; 'check'=''; 'datetime'=(Get-Date -Format 'yyyy-MM-dd HH:mm');
    'result'='Unknown'; 'message'=''; 'data'='';} ) }
2.  $script:lang        = @{}
3.  $script:appSettings = @{}

4.  $script:appSettings['xxx'] = ('')
5.  $script:appSettings['xxx'] = ('')

6.  $script:lang['Error']          = 'Error'
7.  $script:lang['Fail']           = 'Fail'
8.  $script:lang['Manual']         = 'Manual'
9.  $script:lang['Not-Applicable'] = 'N/A'
10. $script:lang['Pass']           = 'Pass'
11. $script:lang['Script-Error']   = 'SCRIPT ERROR'
12. $script:lang['Warning']        = 'Warning'
```

Change lines 4 and 5 to suit your needs. It should contain any settings variables that you may require in your script. Add as many as needed. Next, add the following code the very end of your check, after the closing bracket '`}`' of the function:

```
c-xxx-yy-zzzz -ServerName $env:ComputerName -ResultsPath C:\QA
```

Where `c-xxx-yy-zzzz` is the name of your script function

Option 2

This option doesn't require any extra code, but is slower as you need to compile the entire QA script and wait for it to execute.

To do this, simply follow the instructions in Appendix A for compiling the scripts

Appendix A – Recompiling the QA scripts

Whenever a change is made to the settings file or any of the individual checks, you will need to recompile the QA script.

The reason for compiling into a single file is to make the completed script portable without having 100 separate files all over, potentially being of different versions.

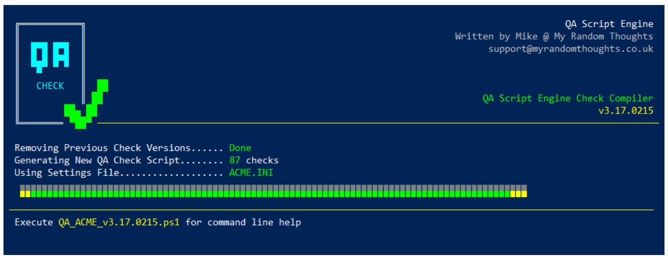
To start, open a normal PowerShell window and change to the folder containing all the script files. Typing the following command will compile all the scripts using the default settings file:

```
    .\compiler.ps1
```

To tell the compile script to use one of your settings files type:

```
    .\comiler.ps1 –Settings {name_of_file}.ini
```

The screen shot below shows the completed process for a settings file called ACME.INI:



```
                                                        QA Script Engine
                                      Written by Mike @ My Random Thoughts
                                             support@myrandomthoughts.co.uk

                                              QA Script Engine Check Compiler
                                                                v3.17.0215

Removing Previous Check Versions...... Done
Generating New QA Check Script........ 87 checks
Using Settings File................... ACME.INI

Execute QA_ACME_v3.17.0215.ps1 for command line help
```

As you can see, the QA script filename contains the short code that is used for this settings file.

Appendix B - Custom Required Functions

There are several functions that are used by various checks, and have been separated out to help reduce bugs. They are listed below…

Check-DomainController

This function returns `$true` or `$false` if the server is a domain controller or not. This is used to determine if local groups should exist or not (c-acc-04-local-groups) or if specific software should be installed (c-sec-13-rsa-authentication).

Check-HyperV

This function checks if the server being checked is a Microsoft Hyper-V Host server. This is used for all Hyper-V-Host (hvh-) checks. This returns a `$true` or `$false` value.

Check-NameSpace

This checks looks to see if a specific WMI namespace exists and returns `$true` or `$false`. This is useful to use before calling a custom namespace or one that may not exist on a particular operating system version.

Check-Port

This function tries to open a connection to a server name on a given port. This returns a `$true` or `$false` value.

Check-Software

This function searches two specific registry keys for the given software name and returns the version number. If no version number exists for the software it returns a version of '0.1'. If the software is not found, then a `$null` value is returned.

Check-TerminalServer

This function checks to see if the Terminal Services namespace exists on the server you are checking. It automatically includes the above `Check-NameSpace` function (the only duplication). This returns a `$true` or `$false` value.

Check-VMware

This function determines if the server is a VMware ESX(i) guest by looking at the serial number value of the server. This returns a `$true` or `$false` value.

Appendix C - Input Types And Validation

Adding comments to the Required-Inputs comment section at the top of your check helps the GUI Configurator inform the user what this particular input is required for. There are some additional controls that can be added to the comments to help shape the input given. These are explained below:

Input Valuation

Most input options will just be a simple string value and therefore will not require any special control codes. Simply add a comment to describe what this input is for. The GUI Configurator will present a single textbox that can be used to enter the details.

Adding one of the following to the end of a comment will provide the input validation shown...

`|AZ`	Letters A-Z only (case insensitive)
`|Numeric`	Numerical validation (integer numbers, as well as floating)
`|Integer`	Integer (whole) only number validation (1, 42, 538)
`|Decimal`	Decimal (floating) number validation (1.23, 42.743, 538.293)
`|Symbol`	Symbols validation (Anything except A-Z or 0-9)
`|File`	File or folder name validation
`|URL`	Web address validation, see below for allowed protocols
`|Email`	Email address validation
`|IPv4`	IPv4 address format validation
`|IPv6`	IPv6 address format validation

Examples

Simple input box

```
Enter your name
```

IPv4 address validation

```
List of DNS IP addresses that you want to check|IPv4
```

Number validation (integers only)

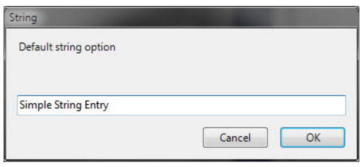
```
Enter the number of days|Integer
```

Input Types

When asking for users to enter data, there are a variety of different input options that can be used. Each one has its own comment control code.

Simple

Without any control code or validation, this is the default input box that a user will see

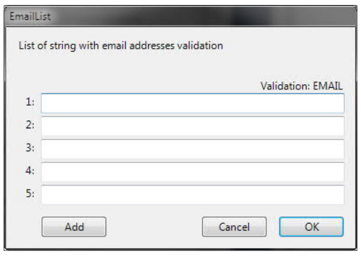

String

Default string option

Simple String Entry

List Of Strings

Adding "List of …" the front of the comment will allow for a list of items to be entered. Shown here is an example with Email address validation

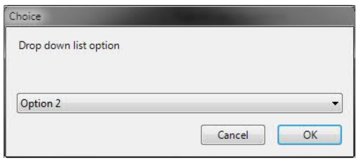


Dropdown List

To have the user select one option from a list of options, add a pipe (|) separated list of items to the front of the comment. Must be separated from the comment with a hyphen (-).

```
"High|Normal|Low" – Email priority
"True|False" – Send email using SSL
"Option 1|Option 2|Option 3|Option 4|…" – Please select an option
```

As many options as required can be listed

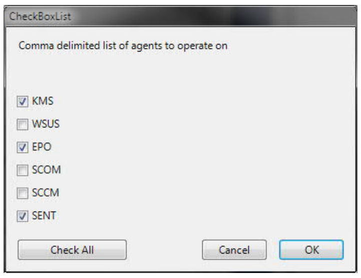


Checkbox List

If more than one option is required, check boxes can be used to choose one or more of the available items. To use, add a comma (,) separated list of items to the front of the comment. Must be separated from the comment with a hyphen (-).

```
"KMS,WSUS,EPO,SCOM,SCCM,SENT" – Select one or more items to install
"Option 1, Option 2, Option 3, Option 4" – Select all that apply
```

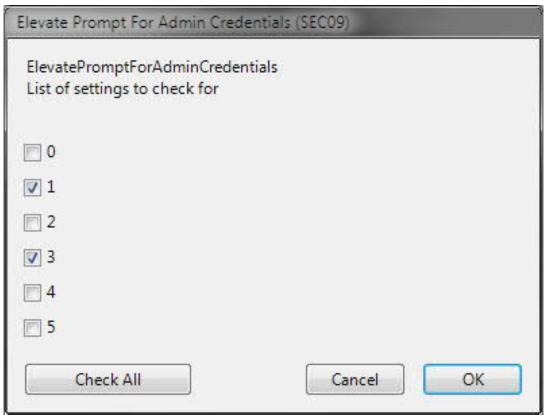
Try to keep the selection list to a maximum of about 10 items



Appendix D - Input Description

When entering data into the GUI front end, there are times when a list of options is presented, but they hold very little information. This optional script header item can be used to add short descriptions to these values.

For example, the check SEC-09 asks for a list of options to choose from when a user is requesting elevated rights. The check looks for one or more of the input values that are in the range 0 to 5. This would mean very little to most people when they are configuring their environment settings.



Elevate Prompt For Admin Credentials (SEC09)

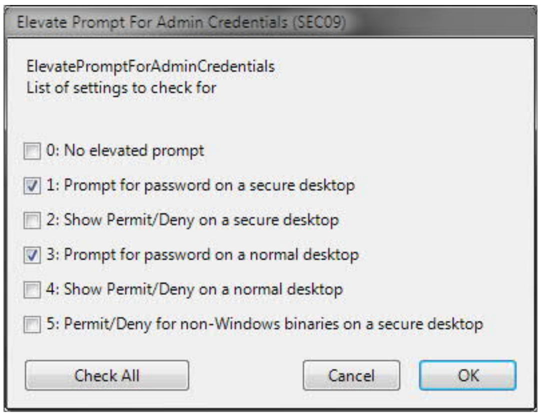
ElevatePromptForAdminCredentials
List of settings to check for

- [ ] 0
- [x] 1
- [ ] 2
- [x] 3
- [ ] 4
- [ ] 5

To add a description to one or more of these items, simply add to your scripts comment header the following information:

```
INPUT-DESCRIPTION:
    0: No elevated prompt
    1: Prompt for password on a secure desktop
    2: Show Permit/Deny on a secure desktop
    3: Prompt for password on a normal desktop
    4: Show Permit/Deny on a normal desktop
    5: Permit/Deny for non-Windows binaries on a secure desktop
```

The format is `{entry}: {description}`, where `entry` is the value that is being asked for.
Once you have added this, the following screen will be shown:



Elevate Prompt For Admin Credentials (SEC09)

ElevatePromptForAdminCredentials
List of settings to check for

- [ ] 0: No elevated prompt
- [x] 1: Prompt for password on a secure desktop
- [ ] 2: Show Permit/Deny on a secure desktop
- [x] 3: Prompt for password on a normal desktop
- [ ] 4: Show Permit/Deny on a normal desktop
- [ ] 5: Permit/Deny for non-Windows binaries on a secure desktop

These descriptions can be added to checkbox selections (as above) or drop down list selections as SEC-15 shows.