New SMART tool: Visualization and Dependency Mapping Tool

*Note: This is a copy from* [*this blog post*](http://blogs.technet.com/b/privatecloud/archive/2014/09/23/new-smart-tool-visualization-and-dependency-mapping-tool.aspx)*. The online version might be more up to date version.*

There is a new kid on the “[SMA Runbook Toolkit](http://blogs.technet.com/b/privatecloud/archive/2014/09/23/updated-sma-runbook-toolkit-smart-download-with-new-visualization-and-dependency-mapping-tool.aspx)” block!

The **Visualization and Dependency Mapping Tool** creates a visual representation of the dependencies of a group of SMA Runbooks : child Runbooks, assets like variables, credentials, certificates and connections. Visualization can happen in the following formats:

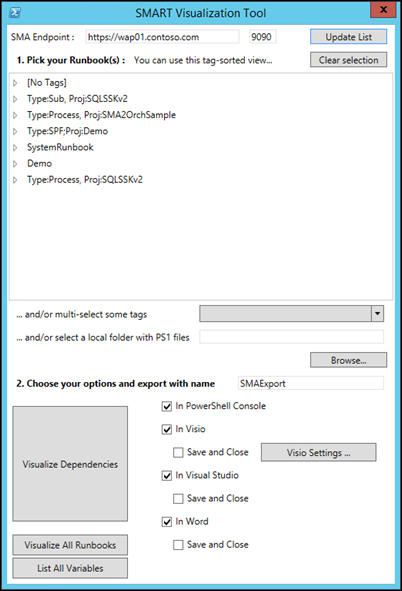
* Text output (in the PowerShell console)
* Visio output
* Word output
* Visual Studio (DGML)

As with [other SMART tools](http://blogs.technet.com/b/privatecloud/archive/2014/09/23/updated-sma-runbook-toolkit-smart-download-with-new-visualization-and-dependency-mapping-tool.aspx), this tool is provided as a sample, and is fully written in PowerShell, so you can easily look at the underlying script and make changes as you see fit. Actual script parsing is achieved using the [Abstract Syntax Tree (AST) technology](http://blogs.msdn.com/b/powershell/archive/2013/10/30/using-asts-with-ise-to-make-scripting-more-productive.aspx).

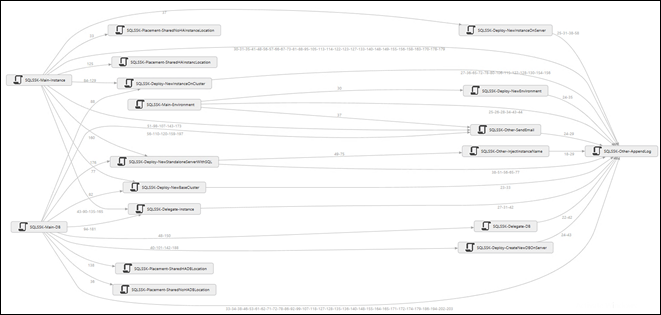
This blog post provides a summary of the features (well…we just did that!), where to get the tool, and how to use it, including requirements. At the end of the post, there are also a few screenshots of sample outputs.

Finally, I wanted to finish the introduction to this post by thanking **Andrew Luty** from the Windows Azure Pack (WAP) team. Andrew has shared some of the DGML routines used within the tool and, as you will see when using it, the output is fast with quite a nice automatic layout!

User interface:

*[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image4.png)*

Visual Studio (DGML) output, each number indicating at which line a Runbook is calling another Runbook :

*[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image47.png)*

## Download location

The tool is provided as part of a single download package including the 3 SMART tools (“Import and Export Tool”, “Documentation and Conversion Helper”).

[DownloadButton](http://gallery.technet.microsoft.com/SMART-Documentation-and-f28fc304)

## How to use the tool

1. **Right click on SMART-VT.ps1 and choose “Run with PowerShell”, or run the script from PowerShell ISE** – The tool should elevate itself to run as administrator if needed. You will need to run the tool with an account that has access to the SMA server  (the tool uses the SMA PowerShell cmdlets under the hood).
2. **In the user interface, update the address and port of your SMA server, to list the corresponding Runbooks and tags**. For Visio visualization, you can also choose different settings. All these settings are automatically kept in a configuration file that the tool will reuse the next time you run it. The XML configuration file is saved in the same folder as the tool itself.
3. **Pick the group of Runbooks for which you want to visualize dependencies**. There are different ways to do so: The main window shows all Runbooks grouped by their same tags. There you can pick a group of Runbooks by selecting a “tag family” , or pick a specific Runbook. Another approach is to select one or multiple tags in the drop-down list that is just below in the user interface. Finally, you can also point to a set of PS1 files on disk – for example the output of the SMART Import and Export Tool. Using the second or third options are the most classic uses of the tool, but you can pick one or a combination of those options, even overlapping choices : The tool will compute the list and make sure to remove duplicates, if any. The Runbooks from local PS1 files could also not be imported in SMA yet, it doesn’t matter to the tool.
4. **You then choose your output options**, and whether you want visualization in the PowerShell console (text output), Visio, Word or a nice graph in DGML (Visual Studio). Visio, Word and Visual Studio should be grayed if they are not detected on the local machines.
5. **Finally, you can click on “Visualize Dependencies”** and let the tool compute the dependencies and display them in the chosen format. If you leave “Save and Close” unchecked for a format, the corresponding application will remain open after “export”. Otherwise, the file is saved, with the name chosen in the user interface.

## Requirements

* The tool should be run with an account that has access to the SMA PowerShell cmdlets against the chosen SMA instance. While you can use an offline copy of the Runbooks, the tool will still try to connect to that instance to display tags, etc.
* Visio output has been tested with Visio 2013 (32-bit)
* Word output has been tested with Word 2013 (32-bit)
* Visual Studio output has been tested with Visual Studio Ultimate 2013
* The tool itself was tested on Windows Server 2012 R2 (64-bit) and Windows Management Framework (WMF) 4.0

## Tips and things you should know about the output

### Performance tips

* **Performance expectations:** When working with “standard size” Runbooks, you should expect a “visualization” performance between 1 and 3 Runbooks per second, the speed increasing with the number of Runbooks. While working with exported local folders may be faster, it is generally not worth it given the short compute time anyway, and the difference tends to decrease as the number of Runbooks increases. When working with large Runbooks, the tool runs faster – about 1 Runbook/second - when querying SMA directly  rather than working with exported files.
* **Which output is the fastest:** Visual Studio visualization is by far the fastest, as the DGML file is dynamically created, and then opened in Visual Studio. In the case of Visio and Word, the applications are opened, and the files are populated on the fly, which takes more time. On this topic, Word output is actually slower than other outputs, and this is due to how PowerShell works with COM automation. There might be some upcoming enhancements on that area in Windows Management Framework (WMF) 5.0, so this may be enhanced in the future.
* Finally, the tool has a **$RunbooksCacheOptimization** parameter which enables a mode where “online” Runbooks are cached to disk before export. Situations where this provides better performance are mostly edge cases, so this option is disabled by default. It is also disabled by default for security reasons : The local cached copy of the “cached” Runbooks – stored in a subfolder where the tool runs – could remain accessible on the local box if the tool is inadvertently closed before it does the expected cleanup.

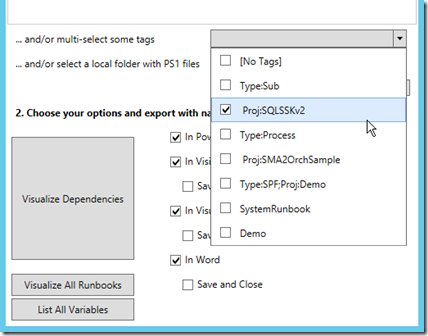
### A few things to know about the outputs

* **Scoping the Runbooks to visualize:** By default, the tool will analyze all Runbooks within the selected scope, and will map their relationships. If some of the Runbooks you selected makes calls to child Runbooks outside of this group, the child Runbooks will still be correctly mentioned in the visualizations, but there are a few limitations: 1/ Child Runbooks (subdependencies and variables/assets not discovered 2/ Visio layout is not optimal for these “exceptions” Runbooks. This is why such child Runbooks will be displayed in red in the Visio and Visual Studio outputs, and mentioned in the PowerShell console. These limitations do not apply if the child Runbooks are part of the group you selected for visualization.
* **Updating your options when running a new visualization:** If you want to run another visualization after a first one, remember you can change each of the selections : You can unselect a given “folder” in the tree view, by clicking on “Clear selection” ; You can uncheck tags in the tags drop-down list ; You can clear the content of the local PS1 folder.
* **Case in the Runbook names:** Runbook calls are case sensitive in the call. So, a Runbook calling a child Runbook named “Invoke-MyRunbook” should be calling its with the same case. If it calls “Invoke-myRunbook” with a non-capital “m”, the two Runbooks will not be fully connected in the output. We thought about making the checks case insensitive, but you can also look at the tool as a way to make sure you have consistency in the way the case is being used throughout the Runbook names. You could still edit the tool’s script to change this, if you want.
* **User interface behavior when computing:** When you click on options like “Update List”, “Visualize Dependencies”, “Visualize All Runbooks” or “List All Variables”, the tool may appear like it’s freezing while it’s computing. That is one of the limitation of doing this via a PowerShell XAML/WPF "GUI" : The tool will not give focus back until the processing has been finished. This is why we tried to add more extensive feedback in the console window at the same time, so you can confirm something is happening. You can move the PowerShell console window around even when the GUI is "waiting". Adding multi-threading, etc. might be possible, but then it may bring more complexity than anything, versus a more "simple" PowerShell script that you can modify to your needs.

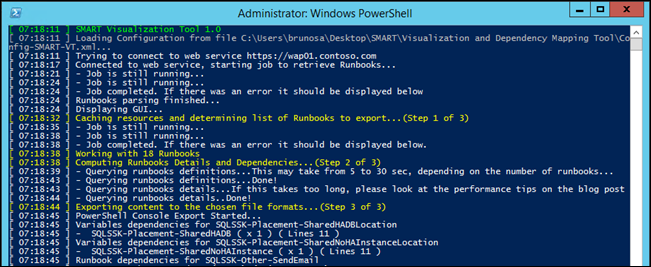
## In closing – a few sample outputs

### Visualizing dependencies across a simple set of Runbooks, by tags

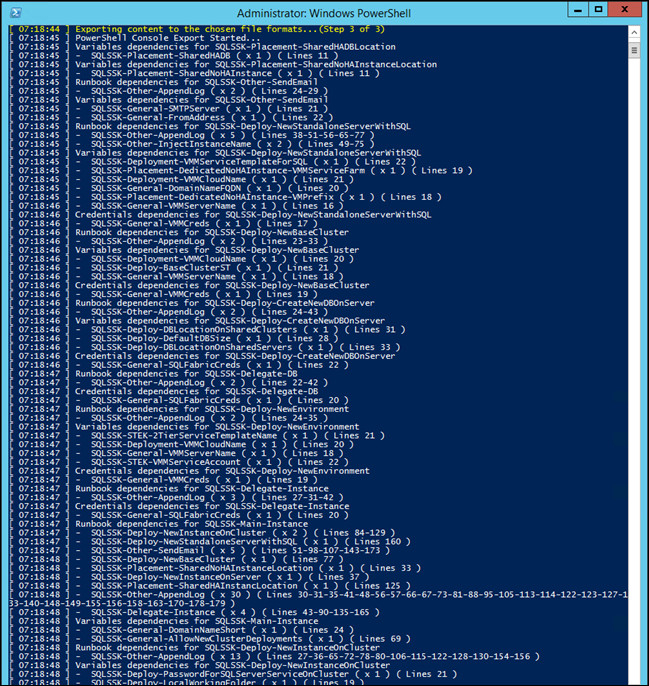
Choosing the right tags in the user interface:

*[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image28.png)*

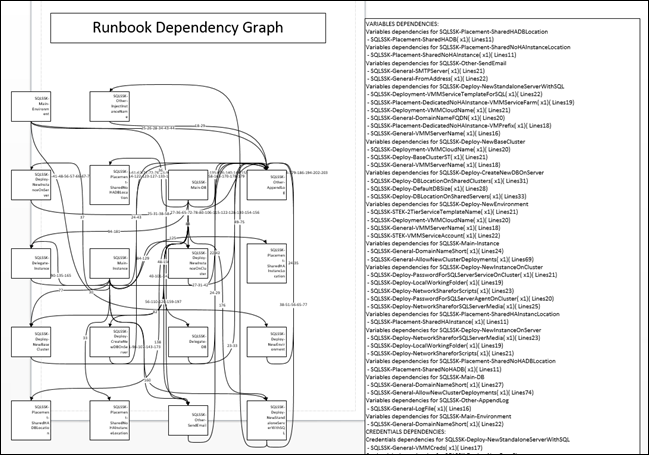
The PowerShell console:

[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image30.png)

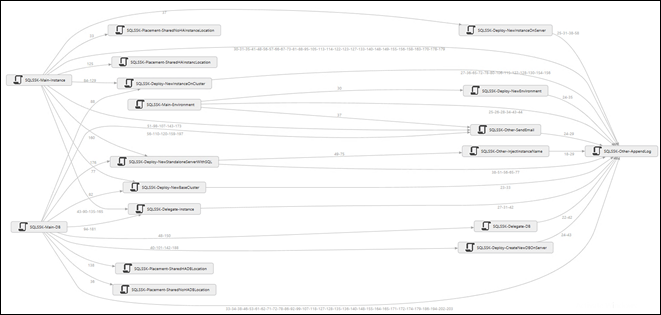
Text output, in PowerShell console (partial output in the screenshot):

[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image35.png)

Visio output – notice the assets being detailed on the right side of the output:

[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image38.png)

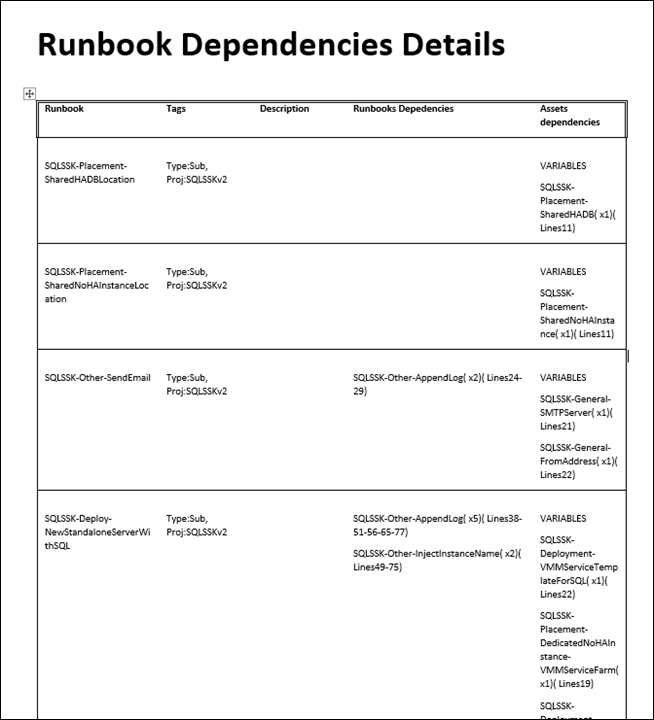
Visual Studio output:

[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image43.png)

Details when hovering on a Runbook show tags assets dependencies:

[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image72.png)

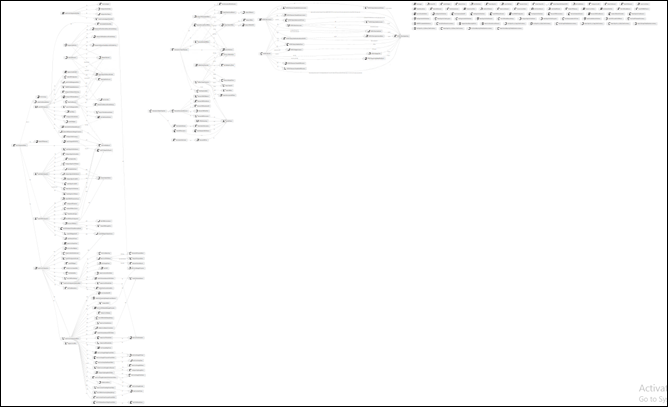
Word output (partial output in the screenshot):

[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image42.png)

### Visualizing dependencies across a massive set of Runbooks

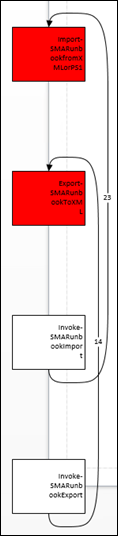
Visual Studio output for 220+ Runbooks – the image is small for both confidentiality and readability purposes *Smile*.

Notice how we can easily spot the two main groups of Runbooks working together on this installation, plus all “standalone” Runbooks on the right

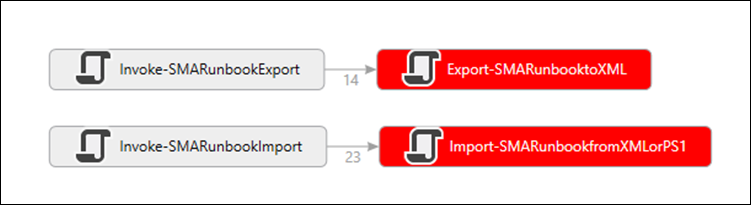
[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image52.png)

### Identifying Runbooks outside of the visualized scope

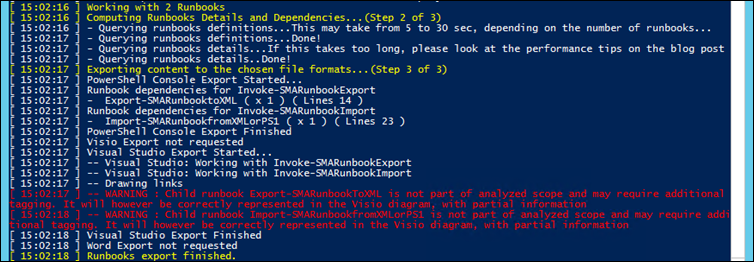
These Runbooks are visible with the red icons in Visio…

[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image63.png)

…and in Visual Studio:

[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image65.png)

The PowerShell console also had an indication about the “orphan” Runbooks:

[](file:///C:\Users\brunosa\AppData\Local\Temp\WindowsLiveWriter1286139640\supfiles31DEC0F0\image67.png)

## Want to learn about other SMART tools?

Here is a [link](http://blogs.technet.com/b/privatecloud/archive/2014/09/23/updated-sma-runbook-toolkit-smart-download-with-new-visualization-and-dependency-mapping-tool.aspx) to a blog post summarizing them!

The current SMART package includes this tool, plus the “Import and Export Tool” and “Documentation and Conversion Helper”.