# Appendix B: Example from Blog – Part 1: Example Runbook

**Chapter 1…it all began on a warm late-summer afternoon…**

A couple of months ago - during my attempt to better understand the processes used in **Orchestrator Runbook Design** (*and to ensure I did not have to spend hours of time creating documentation for runbooks that I would eventually design*) - I began researching utilities and/or scripts I could use to automate the “documentation” process for a particular runbook.

*<Rant>*

*I mean – really – I just spent x days/weeks creating and designing the whole thing – who has the mental stamina to go back to the* ***very beginning and then re-live those moments over again back to back****! Or – on the flip side – who would want to spend the extra weeks* ***designing the runbooks while pausing for periods of 2-5 minutes in between runbook design steps in order to take snapshots, add comments into a Word document, etc.***

*AND THEN… try to return back to your design - only to realize you can’t remember that perfect solution for problem X you had figured out - because you were so focused on documenting the stupid* ***“Initialize Data”*** *activity!???!!*

*</Rant>*

That’s when I happened to stumble upon the [S.M.A.R.T Documentation and Conversion Helper 2.0 Toolkit](http://blogs.technet.com/b/privatecloud/archive/2014/05/08/updated-tool-smart-documentation-and-conversion-helper-for-your-orchestrator-runbooks.aspx) (*Formerly the Orchestrator Documentation Script*) released on this blog.

I absolutely ***loved*** the concept and was very excited when I first downloaded and ran it against my Orchestrator Lab environment. Besides – the fact that it was created completely in my favorite go-to administration tool – **PowerShell** – made it doubly cool.

**Short-Lived Romance**

**BUT -** I have to admit – while I can see how the tool had very valuable use for instances where it came to converting runbooks to SMA runbooks, etc., I was let down by the resulting documentation in the end.

“*But Why?*”, do you ask? Most of the information you need to understand the runbooks and how they operate are there, and the links even follow the proper workflow paths that your runbooks use. “*What...*” might you ask, “*…is the problem good sir??!”*

The problem, you see – is that I want to be able to provide a set of documentation/reports/information for my runbooks that allow **not just someone who has gone to a two-week boot-camp on System Center Orchestrator 2012 R2 Runbook Design can understand if they need to know how a runbook works**.

I want an especially-adept Operator or a versatile Systems Administrator (*i.e. grasps basics regarding systems automation, workflows, basic scripting language and theory, etc*.) to be able to “go through” the documentation and come out with the information they need to do “standard checks”.

After two months of tinkering, I believe I have come to a point where I can at least share what I’ve learned with the community. ***I hope someone as curious as myself can take this and solidify it to something worthy of the TechNet Script Repository, and maybe someday it will become a staple of many more SCOrch Admin’s toolkits other than my own.***

**Examples of my final resulting script will be shown later as we walk through documenting the following runbook:**

**Note:** The example runbook we use in this blog is an attempt to demonstrate how different activities, links, etc. are documented from the script. The workflow itself is not as efficient as it should be – but that’s by design!

**Example Runbook – Creating Files and Logging the Info**

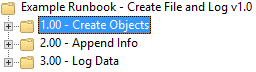
Before we get to the good part – the script itself – I want to provide a basic runbook we can use to generate some docs and so I can include as an example of the results I was able to generate using the included **ExampleRunbookv1.0.OIS\_Export** file.

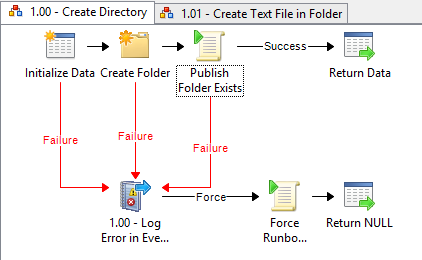
This blog focuses on the design of this example runbook and as a quick reference for anyone interested in why I did some of the things I did. For instance – I KNOW these runbooks can be condensed quite a bit to streamline the workflow…but by doing it this way I tried to provide a demo that included the versatility of the script (*i.e. the ability to process multiple runbooks within a folder, etc.*)

The design for this runbook is intended to be simple, yet contain enough complexity that you can use your own imagination to extrapolate how you can extend these concepts into your own runbook design strategy.

So let’s get down to business and walk through the design structure for the runbook example:

1. **– Create Directory**

 **Runbook Location** (*above-circled red):*  Folder that runbook **1.00- Create Directory** resides



**1.00– Create Directory Runbook** *(above)*

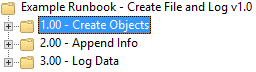
This simple runbook takes a single parameter (***DirectoryPath***) and will attempt to create the directory using the default runbook service credentials. It will then return any information we provide from the local data bus to the master runbook’s data bus.

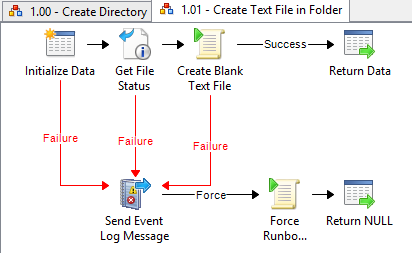
If the activity creating the folder fails, an event will be logged in the servers Event Log, and then a PowerShell script runs that will force the runbook to fail (*otherwise, the “****Log Event****”**activity can return a “success” and will be interpreted incorrectly by the runbook for the rest of the workflow*).

**The data returned from this runbook is as follows:**

|  |  |  |
| --- | --- | --- |
| Variable | Type | Description |
| WasFolderCreated | Boolean | True/False if the folder was created |

**1.01 – Create Text File in Folder**

 **Runbook Location** (*above-circled red):*  Folder that runbook **1.01-Create Text File in Folder** resides



**Example** *(above):* **1.01-Create Text File in Folder**

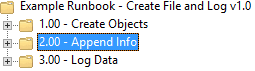
This simple runbook takes two parameters (***FileRootFolder*** *and* ***TextFileName***) and will attempt to create a blank text file using the default runbook service credentials. It will then return any information we provide from the data bus.

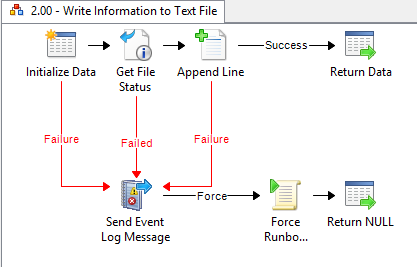
If the activity creating the text file fails, an event will be logged in the servers Event Log, and then a PowerShell script runs that will force the runbook to fail (*otherwise, the “****Log Event****”**activity can return a “success” and will be interpreted incorrectly by the runbook for the rest of the workflow*).

**The data returned from this runbook is as follows:**

|  |  |  |
| --- | --- | --- |
| Variable | Type | Description |
| FileCreated | Boolean | True/False if the file was created |
| FilePath | String | The path of the file that existed |

**2.00 – Write Information to Text File**

 **Runbook Location** (*above-circled red):*  Folder that runbook **2.00-Create Text File in Folder** resides

 **Example** *(above):* **2.00-Write Information to Text File**

This simple runbook checks for the existence of a file. If the file exists, then we will add the line:

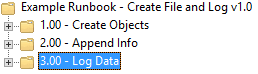
**“[ Example Runbook v1.0 Test ]: Successfully Completed Append Test**!”

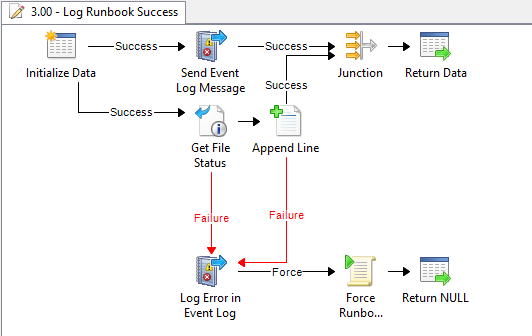
If the activity that appends data to the text file fails, or if the file does not exist, then an event will be logged in the servers Event Log, and then a PowerShell script runs that will force the runbook to fail (*otherwise, the “****Log Event****”**activity can return a “success” and will be interpreted incorrectly by the runbook for the rest of the workflow*).

**The data returned from this runbook is as follows:**

|  |  |  |
| --- | --- | --- |
| Variable | Type | Description |
| FileExisted | Boolean | True/False if the file existed |
| AppendSucceeded | Boolean | True/False if the file had data appended |

**3.00 – Log Runbook Success**

 **Runbook Location** (*above-circled red):*  Folder that runbook **3.00-Log Runbook Success** resides

 **Example** *(above):* **3.00-Log Runbook Success**

This simple runbook checks for the existence of a file. If the file exists, then we will add the line:

**“[ Example Runbook v1.0 Test ]: Successfully Completed Append Test**!”

If the activity that appends data to the text file fails, or if the file does not exist, then an event will be logged in the servers Event Log, and then a PowerShell script runs that will force the runbook to fail (*otherwise, the “****Log Event****”**activity can return a “success” and will be interpreted incorrectly by the runbook for the rest of the workflow*).

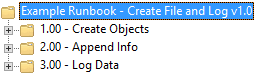
**The data returned from this runbook is as follows:**

|  |  |  |
| --- | --- | --- |
| Variable | Type | Description |
| FileExisted | Boolean | True/False if the file existed |
| AppendSucceeded | Boolean | True/False if the file had data appended |

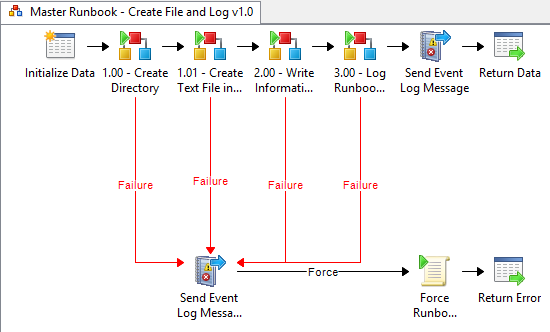
**Master Runbook**

The **Master Runbook** we will call “**Example Runbook – Create File and Log v1.0**”

This runbook will be a runbook that is comprised solely of “**Invoke Runbook**” activities () that link the following sub-runbooks together to create the whole workflow:

  
**Runbook Location** (*above-circled red):*  Folder that runbook **Master Runbook – Create File and Log v1.0** resides

0



**Example Master Runbook** *(above)*: **Create File and Log v1.0**

This is the master runbook that controls the runbooks we’ve been looking at to this point. It basically just checks the success/failure of each runbook and will send event log messages that also depend on the success/failure in question.

If one of the runbooks fail, then an event will be logged in the servers Event Log, and then a PowerShell script runs that will force the master runbook to fail (*otherwise, the “****Log Event****”**activity can return a “success” and will be interpreted incorrectly by the runbook for the rest of the workflow*).

**The data returned from this runbook is as follows:**

|  |  |  |
| --- | --- | --- |
| Variable | Type | Description |
| Status | Boolean | True/False if we made it to 2.00 and it succeeded (yes – we’re skipping 3.00 because it’s extra logging and didn’t effect the original intent of the runbook) |

**SCOrch Tip of the Day: Making Link Activities Easier to Understand**

When creating the link objects, you should include the logic you use in the **description** are of the link. This way – it will be easier for people to read your resulting documentation. The script DOES provide a method for you to see all the link conditions *(see second example below)*, but anyone unfamiliar with basic programming theory will have a difficult time translating the way we convert the values by default from within the tool (*sorry – didn’t want to take the time to modify*).

  
**Example** (*above*): Add a description to the links as you create them to be easier to understand the conditional logic used later when you are troubleshooting.

  
**Example** (*above*): How the script will present conditional “Triggers” found on **Link Activities.**(*Note that his is derived from the runbook export file, not the description field*)

**Example Downloads: Runbook and Resulting Reports**

Download the following zip to see the exported runbook I used, as well as the resulting HTML reports. **The next post will walk through the process I used to create the reports you can download below:**

Click here to download my example runbook and report files

When you unzip the reports, the root folder itself doesn’t matter, but keep all the runbook export folders in the same directory for the link structure to work properly. Also, you may get a warning about running some content in IE (because of my JQuery code) – so just **click** “**Allow blocked content**” when asked:

