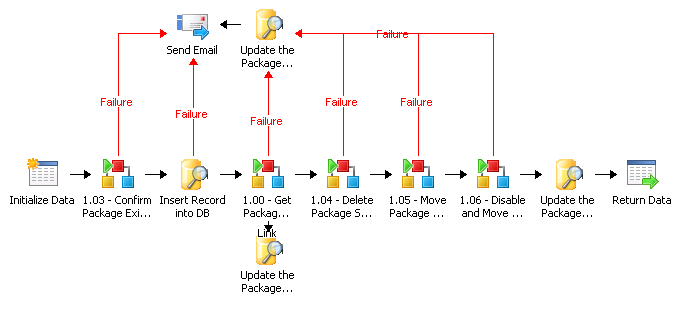
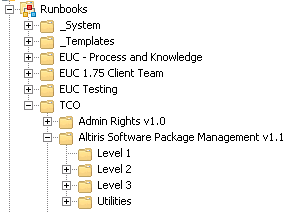
**Microsoft Orchestrator**

**Runbook Design Best Practices**



1. **Folder Structure**

The folder structure will be designed with a structure of Runbooks\Group\Project\Level 1, Level 2, Level 3, etc. The Level X folders are subfolders for a given project that link workflows from a higher level to their lower level (more detailed) runbooks. The main project level contains workflows that are triggered externally to start the automation process. The Shared folder is used for common /utility runbooks that are called from different levels.



The project folder name should be versioned (e.g. v1.0)

This same structure should be followed within the Global Settings folders however the Level X subfolders should not be used.

The “\_Templates” folder is used to store runbooks that are generic and reusable across runbook projects.

The “\_System” folder is used to store runbooks that are used for back end server management, typically executed and managed by the system administrators.

There should be no runbooks at the root “Runbooks” level. All runbooks should be within a sub folder.

1. **Naming Conventions**

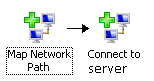
Runbook names should be a 1-4 word description of their primary function. Sub-runbooks within a lower level should be numbered to match the level and item (for example, the first runbook under Level 1 would be 1.00 - <name> and the second would be 1.01 - <name>, etc)



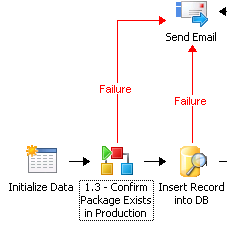
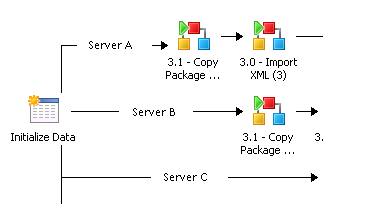
Runbooks that start with a monitor or trigger need to be named “Monitor - <text>” to indicate they are monitor runbooks and should always be running and also if a server needs to be reboot, these runbooks can be stopped and re-run on a different runbook server without any issues.

There are no specific naming conventions for Counters, Variables or Schedules.

Actions should be renamed to be a short 1-6 word description of what action is being performed or on what target the action is being performed on. For example, the “Map Network Path” action should be renamed to show which server it is connecting to.



Link names should only be modified from the default of “Link” if there the link is used in a logic selection or a parallel execution.

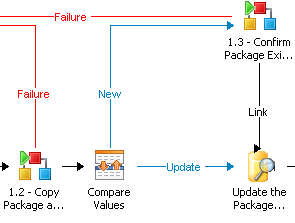


1. **Link Layout and Colors**

The placement of Actions should be done to keep link lines as straight as possible. Limit overlapping link lines as much as possible.

Color links with the following scheme:

* Black=Standard / Success – includes parallel processing
* Red = Failure
* Turquois = Selection
* Orange = Expected to fire multiple times



Links should only be colored if there are separate success (black) and failure(red) paths or if a selection is made between links (turquois).

Avoid using long link trigger delays as this can result in the job appearing to be hung from the operator console. Leverage sleep.exe or a .Net Run Script with PowerShell to perform the delay so that it is visible within the console.

Note that if a runbook does not run through to completion because of conditional checking on a link, the runbook itself will return a value of success. You must use the Generate Error custom action to have the runbook itself return an error.

Exclude filters on links take priority over include filters. This means that if any exclude filter condition is satisfied then the link will not be followed no matter how many include filter conditions are satisfied.

1. **Variables**

Server names, email addresses and folder shares should never be hard coded but should always be referenced as a variable within the Global Settings section of the Runbook Designer.

Environment variables can be referenced by creating an Orchestrator variable to is set to a value of %VARIABLENAME%. For example, you can set an Orchestrator variable equal to %TEMP% and then when the Orchestrator variable is referenced it will be dynamically set to value of TEMP on the executing Runbook server. The environment variable is resolved on the Runbook Server where the policy is running and is not case sensitive. Only system variables can be resolved; user variables will not be resolved. If the environment variable does not exist, the text specified within the variable will be returned literally. See <http://technet.microsoft.com/en-us/library/hh440543.aspx>.

NOW() returns the current time on the Runbook server. You can pass arguments to this function to return specific portions of the date or time. For example, NOW(hour) will return the current hour. The following are the valid arguments for the NOW() function: day, dayofweek, dayofyear, month, year, hour, minute, second, millisecond.

1. **Counters**

Note that counters are shared between all runbooks. Use caution when using counters across runbooks that execute with more than 1 instance as one runbook may reset a counter value that would impact another running runbook. When testing runbooks within the debugger the current counter value will be pulled however any changes to counters will be discarded. For counters that are specific to a running runbook instance, consider using a PowerShell variable within the Run .Net Script action as this variable is local to the running runbook instance and is discarded when the runbook completes.

1. **Runbook Naming Convention**

Top level Runbooks:

Description of the high level task they perform. <Verb> - <Noun>

Example: Disable Global Package

Sub Runbooks:

<Number> < Verb> < Noun>

Example: 1.00 – Get Package Details

1. **Runbook Concurrency**

Runbooks should be configured to have a maximum job concurrency set of 20 or less.

In a scenario with a single runbook server and nested runbooks there can be a deadlock if the maximum runbooks have been reached but individual runbooks cannot finish because all running runbooks are wanting to launch another child runbook in order to finish but none of them can because the maximum number of running runbooks has already been reached. With multiple runbook servers this is less of an issue because the secondary runbook server can be monitored and as jobs spill onto this server then you know that an additional runbook server needs to be installed or the max concurrency of the first server needs to be increased. Either way there will not be a deadlock.

1. **Runbook Size**

Runbooks should contain no more than 30 items. Runbooks of this size or larger should be broken into small sub-runbooks.

1. **Runbook Design**

Every runbook should start with “Initialize Data” and end with a “Return Data” even if it is not expected to be called from another runbook.

Avoid using alternate icons for individual actions because they become part of the data stored in the database and can slow down runbook execution.

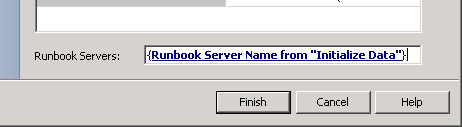
When flattening variable data, use ;;DELIM;; as the delimiter to ensure that it does not exist within the data provided.

When using parallel processing with cloned actions, ensure the action icon names are unique so that they can be differentiated from each other when debugging. Also, when using the custom Generate Error action, ensure a unique action is used per process flow to allow visible debugging in the console to determine which flow generated the error.

If two runbooks are 90% the same, consider consolidating them into a single runbook with an additional parameter within the Initialize Data action to allow it to do different actions depending on the scenario.

Design runbooks so that they are generic and easily extendable and reusable.

Consider that child runbooks may be launched by a different runbook server. This can be an issue where a previous runbook mapped a network drive and that drive mapping is not present on the other runbook server. To allow the parent runbook to run on any free runbook server but enforce that any child runbooks run on the same runbook server you can pass in the runbook server name from the databus whenever a child runbook is invoked.



1. **Runbook Documentation**

Top level runbooks will have the Properties – General – Description field filled out with a general description of the purpose and function of the runbook. If the runbook has any perquisites or assumed conditions then they should be documented here as well.

1. **Runbook Server Assignments**

Avoid assigning runbooks to execution on specific runbook servers as this will cause them to not fail over to an alternate runbook server in the event of a failure of the assigned runbook server. Also the jobs will not spill over to an alternate runbook server if the assigned runbook server is already running the maximum available concurrent jobs. In this scenario the job would be queued up rather than run on an available runbook server unless additional runbook servers are selected. Also assigning specific runbook servers can cause problems when trying to update a specific runbook server with a new IP or to reboot it.

1. **Runbook Linking**

In order to properly propagate the success or failure of invoked runbooks, the link leaving an Invoke Runbook should only continue if the “Child runbook status” equals “success”.



To catch failures on executing a link use you need to catch all non-success conditions.



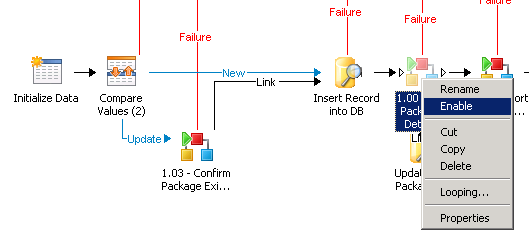
Use the Generate Error custom action for child runbooks to ensure that they generate an error if their child runbook errored so that the calling parent runbook will also be notified of the error condition.

Include the name of the invoked runbook in the description of the Invoke Runbook action that calls the sub runbook.

1. **Runbook Testing**

The Runbook Tester can be used to test individual runbooks however it will run the runbook use the account of the logged in user and NOT the process account used by Orchestrator. One exception is any child runbooks that are invoked by the runbook being tested are run by the process account.

Another strategy for testing is to disable an action within runbook to limit the runbook from running through to completion and allow testing of individual components of the runbook.



1. **Runbook Actions**

**Use built in actions rather than scripting if possible.**

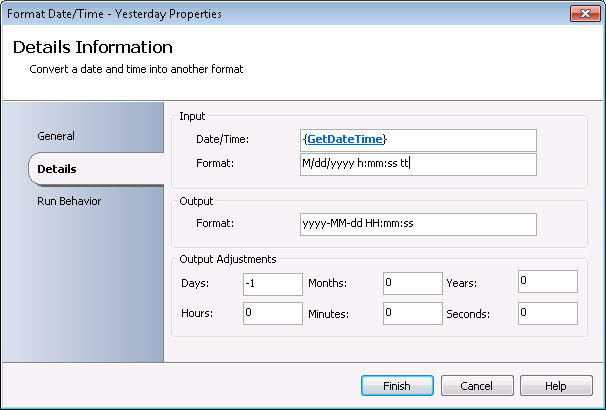
**Append Line** – This action causes problems with jobs that allow more than 1 runbook to run simultaneously because the action puts a lock on the file so concurrent attempts to append the file will result in an error. Consider using high retry counts within a loop.

**Delete File** – This action fails if the specified file has the read-only flag set.

**Delete Folder** – This action fails if the specified folder contains any files with the read-only attribute set.

**Disconnect Network Path** – This action should never be used within runbooks that have a maximum number of concurrently running jobs set higher than 1. The reason is that network connections are shared by the runbooks since they exist within the session of the process account. If two jobs are running simultaneous but at different start points, one could disconnect the network path that is still needed by another running runbook.

**Format Date/Time –** When referencing the Now() method, the date time is returned in the formation of M/dd/yyyy h:mm:ss tt. Use the Format Date/Time activity to convert this format to the format expected for the system you are interacting with. The example below converts the format for use within SQL.

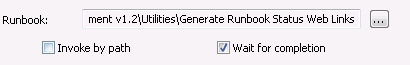


**Invoke Runbook** – By default the child runbook can be spawned on any runbook server. This is determined based on the current assigned primary server, the server availability and the number concurrent jobs currently being run. If there is a dependency for a child runbook to run on the same server as the parent runbook (e.g. for drive mapping or file processing reasons) you should use specify the runbook server name dynamically to match the runbook server used by the parent runbook. Avoid doing this as much as possible because it limits the ability for the runbooks to switch to a server with more capacity if the current runbook server has reached is maximum job concurrency level.



If runbook needs to call itself recursively, the check box for “Wait for completion.” must be unchecked otherwise the runbook will hang trying to call itself.

Using the Invoke Runbook action causes the runbook server to start a new policymodule.exe which adds about 5 seconds to the runbook execution. This needs to be considered when reading large amounts of data from a database or a spreadsheet where the data is not flattened and each line/record is processed. For example, if a 4000 line spreadsheet file is read and each line uses Invoke Runbook this would take approximately 5 ½ hours to complete.

Avoid using the “Invoke by path” checkbox. This will cause the Invoke Runbook to fail when the name of the parent folder is updated with a new version name.

**Invoke Web Service** – Use [username@domain.com](mailto:username@domain.com) for the username within the Security tab when cross domain authentication is needed (e.g. username is in the NA domain but the web server is in the LA domain).

When embedding SQL queries into XML files there are some characters that must first be converted:

### XML character references

Unlike traditional HTML with its large range of character entity references, in [XML](http://en.wikipedia.org/wiki/XML) there are only five predefined character entity references. These are used to escape characters that are markup sensitive in certain contexts:[[7]](http://en.wikipedia.org/wiki/Character_encodings_in_HTML#cite_note-6)

* &amp; → & ([ampersand](http://en.wikipedia.org/wiki/Ampersand), U+0026)
* &lt; → < (less-than sign, U+003C)
* &gt; → > (greater-than sign, U+003E)
* &quot; → " (quotation mark, U+0022)
* &apos; → ' (apostrophe, U+0027)

**Junction** – When using a junction with multiple parallel running actions, if any action fails the entire runbook will return a value of “warning” even if the runbook completes due to other actions running successfully.

If the object specified within the “Return data from” is executed multiple times then the Junction action will fire multiple times. To resolve this issue, change the “Return data from” to be equal to <None>. Note however if that if the value is set to <None> then no data from actions prior to the Junction will be available on the databus.

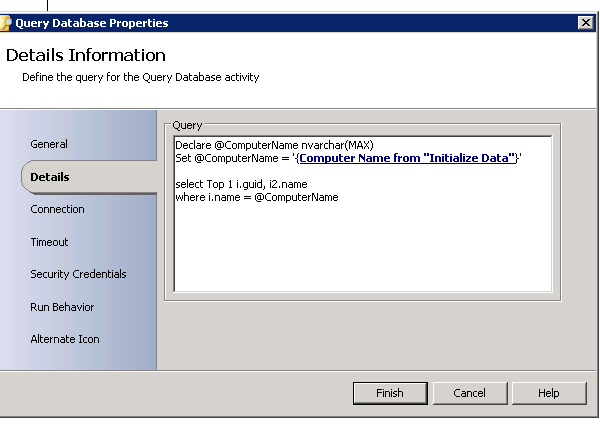


**Map Network Path** – This action will fail if the path being mapped is already mapped by another user on the system. For example if an administrator remote desktops into the system and browses a network path that is used by this action within a runbook then this action will fail when it tries to map this path using the process account credentials. Also ensure that connections names used across actions are the same. For example, if you map to [\\server.domain.com\packages](file://server.domain.com/packages) and then attempt to copy a file from [\\server\packages](file://server/packages) it will fail.

Map directly to the IPC$ share rather than a specific drive letter so that you can connect to multiple drive letters or alternate shares without having to call this activity multiple times.

**Query Database** – Ensure that you consider the source of the data used to perform SQL queries to make sure you protect against SQL injection. <http://en.wikipedia.org/wiki/SQL_injection>

Any data passed into a SQL query from an untrusted source must be configured as a parameter using the DECLARE and SET commands.



If the data you are querying could contain the delimiter value ; and you are parsing out values based on the ; then you will need to use the SQL REPLACE command to replace the ; characters with a comma or some other value in order to allow properly parse the returned data from SQL. Below is an example:

SELECT REPLACE([Test],';','\_')   
FROM [OrchestratorCGLAdmin\_prod].[dbo].[Test\_Table]

**Query XML** – This object leverages XPath queries to extract data from XML files. Review <http://www.w3schools.com/xpath/xpath_syntax.asp> for more information. Note that XPath queries are case sensitive.

Examples:

|  |  |
| --- | --- |
| **Expression** | **Description** |
| nodename | Selects all child nodes of the named node |
| / | Selects from the root node |
| // | Selects nodes in the document from the current node that match the selection no matter where they are |
| . | Selects the current node |
| .. | Selects the parent of the current node |
| @ | Selects attributes |

**Run .Net Script**  -

* When using action with a PowerShell script, do not use the Exit command as it will result in Publish Data variables not returning any values .
* When publishing multiple variables that are arrays, the array sizes will be aligned and the smaller array will be extended with blank values to match the larger array.
* If you are using the Field data manipulation function within the script body but the activity where input data is pulled from did not execute then you will receive the error message “Error initializing extension”.
* When reading unstructured data into variables that contain single or double quotes, use the following syntax to avoid script errors. Using @’ to avoid $variables from being resolved. Use @” to automatically resolve variables:

$Variable = @’  
 I wish this line didn’t contains any “double” quotes.  
‘@

* Validate the data being passed to the runbook using the following regular expression to ensure that Orchestrator does not run unauthorized PowerShell commands.

^(([?!@('|")).)\*$](mailto:?!@('|%22)).)*$) #Only matches to strings that don’t include @’ or @”.

* "Cannot invoke this function because the current host does not implement it." Is generated from the Run .Net Script action when a powershell command is executed that expects input from the user. This can be resolved by modifying the powershell script command to no longer prompt the user for input.
* Capture error line using the following syntax:

try{

}  
catch{ throw("$\_ at line number $($\_.InvocationInfo.ScriptLineNumber).")   
}

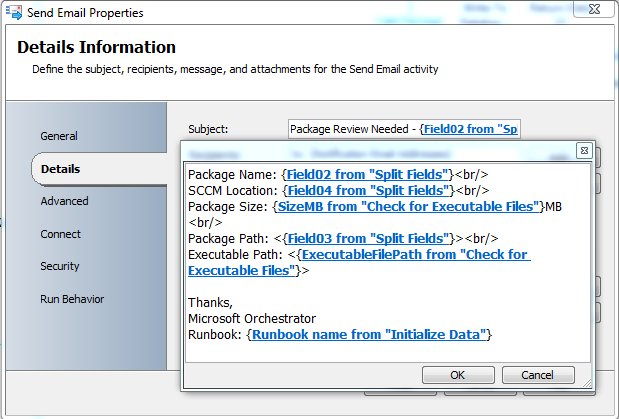
Converts:  
Exception setting "ProgramFlags": "Cannot convert value "-1995398143" to type "System.UInt32". Error: "Value was either too large or too small for a UInt32.""

To:  
Exception setting "ProgramFlags": "Cannot convert value "-1995398143" to type "System.UInt32". Error: "Value was either too large or too small for a UInt32."" at line number 151.

**Run Program –** The Run Program activity action leverages the Orchestrator Remoting Service on the Runbook Server which in turn calls opexesvc.exe which is actually a modified version of psexec.exe to remotely do the program execution on the remote system. Interactive mode is not supported on Windows 7 and 2008 systems. The Security Credentials tab is used to define the credentials used by the opexesvc.exe to connect to the remote computers \admin$ share. The Run As credentials within the Advanced tab are used to configure the user account that the opexecsvc.exe should run the specified command as.

**Restart System**- This action requires a minimum of a 30 second wait period. It does not hold the runbook, it will initiate the timer and allow the runbook to continue. All users on the system, including those connected through RDP on a server will receive a dialog box prompt letting them know that a reboot has been initiated. The user can close the dialog window but they cannot stop the reboot.

**Send Email** – Use a standard email address for all emails generated from Microsoft Orchestrator (e.g. [donotreply@company.com](mailto:donotreply@company.com)). Within the email identify that it was sent from Microsoft Orchestrator and that the recipient should not reply to the email address as it is not monitored. Also include the name of the runbook that generated the email.

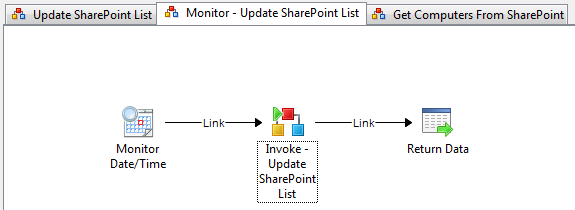


1. **Runbook Monitor jobs**

To monitor a database you can use the Query Database action with a loop and exit based on the query result or you can use a Monitor Date/Time action to periodically query a database and process the results accordingly. This can work well for basic lookups, for more advanced processing a stored procedure may need to be written on the database that is then called from the runbook itself.

Note that jobs that start with a Monitor action never exit on their own. When the monitor is triggered, the job will spawn a clone of itself to run the rest of the runbook actions and the original Monitor runbook will continue running and wait for the next trigger event.

Monitor runbooks should consist of the trigger and a call to invoke the main runbook being monitored. Keeping these separate allows for cleaner runbooks and for the main runbook to be updated without having to stop and restart the monitor runbook.



Monitor runbooks that are triggered should be very short and typically not actually do any processing but instead immediately call a child runbook which does the primary processing. This can avoid problems in scenarios where the monitor runbook is triggered thousands of times (e.g. SCOM event monitoring).

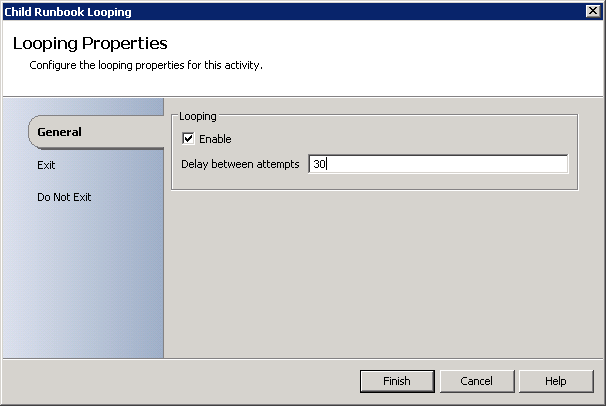
Considering setting Properties – Events – “Report when the Runbook runs for more than: <second>” to report when a monitor runbook is running longer than the execution interval and looping on top of itself.

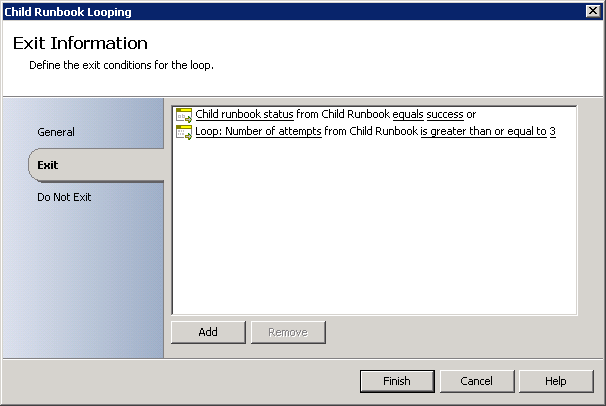
1. **Runbook Looping**

Consider enabling looping within top level runbooks or within individual child runbooks that are generating sporadic errors. This can reduce the amount of troubleshooting and re-work for a failed runbook that would have worked successfully if another attempt was made.

If runbook looping is enabled, the logic within the loop will happen before any links are followed even if the same conditional logic exists within the loop and the link.

The conditions within the Do Not Exit properties of the loop will take precedence over the Exit conditions. This can result in infinite loops so care must be taken to ensure they are configured correctly.

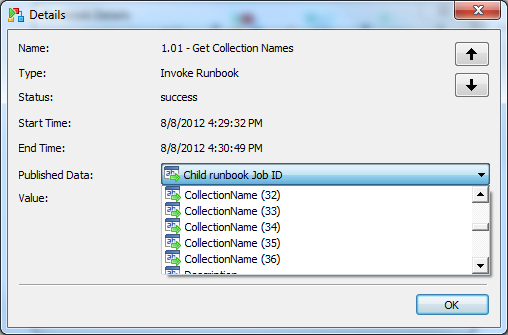
Configuration for looping on an Invoke Runbook action. 



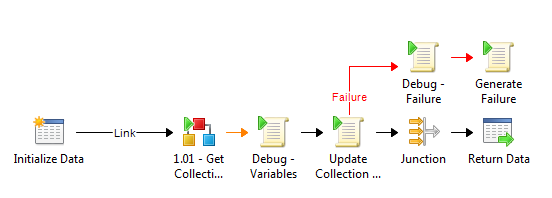
1. **Multiple activity executions**

Many activities support being executing multiple times. For example, by default the Query Database activity will call every activity after it for each row it returns. In this scenario it is difficult to troubleshoot a runbook failure when an activity further down the runbook generates the error but this activity was called multiple times (once for each row in the Query Database activity).

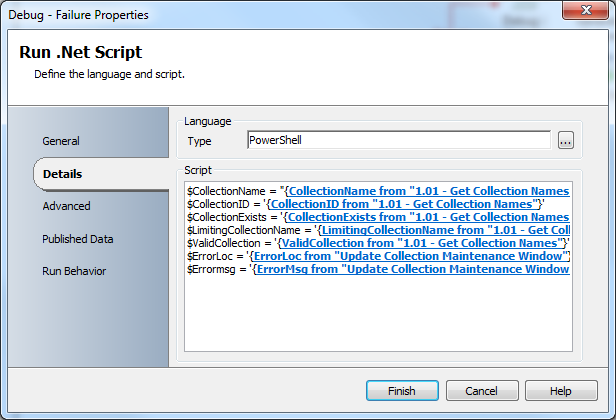
As an example, you can see that 36 different CollectionName values were returned.



To help facilitate quicker debugging, you can add a PowerShell script that simply return all of the variables in the current execution.



This is especially useful before calling a Generate Failure activity where you can place all of the relevant data on the database to determine what data was passed to the activity that generated the failure. You could then write this to a log file or a database for future review.



1. **Stopping a running runbook**

Note that when stopping a running runbook it does not stop any child runbooks that were spawned. You must manually track down any running child runbook and stop them as well.

1. **Scripting languages**

Run .Net Script will leverage PowerShell by default.

1. **External Scripts**

Calling external scripts should not be used. Scripts should be imported into the Orchestrator Designer and managed there to provide source control, auditing and ensure that a script is not modified outside of the runbook which would allow activities to run with elevated privileges unintentionally. If external scripts are used they need to be moved to a secure location to ensure they are not modified.

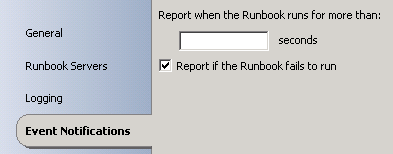
1. **Integration Packs**

Integration packs will be developed using VB .Net

1. **Logging**

“Store Activity-specific Returned Data” and “Store Common Returned Data” settings within each runbook should be configured differently for the test environment versus the production environment. You should not have these enable in production as they will store a lot of extra data in the database. Instead use them during dev and QA. Use only an external system to log in production, if needed.

Top level runbooks should have the Event Notifications option configured to report if the policy fails to run. This will generate an entry in the Orchestrator Events log.



Logging to a database, file or email should include the runbook ID to allow for quicker debugging by being able to go to the specific runbook that failed.

Server log purging should be enabled and should fairly aggressive in your production environment. For example, keep only the last 2 days of execution data. Orchestrator has poorly written internal queries that do not perform well when internal database tables grow large.

1. **Reporting Database**

A separate database should be used for the storage and reporting of policy specific information. Every runbook should have an ROI value per runbook execution and each execution should be logged so that runbook ROI metrics can be collected.

1. **Segregation of Duties**

Orchestrator environments will be segregated into Development, QA/Staging and Production environments. A central team will manage the QA/Staging and Production environments.

1. **Development Environments**

External teams will each have their own development (dev) environments where they will have full rights to create and modify runbooks. Dev environments will run their services using a low privilege process account that is different from that which is used in production. There is no security to the configuration options within the Runbook Designer console which requires full separate dev environments per team.

1. **QA/Staging Environment**

The QA/Staging Environment will be configured to identically match the Production environment. This includes running their services with the same high privilege domain process account.

New runbooks will be kept in the QA/Staging environment for 1 week for testing to ensure everything is working properly before moving to the Production environment.

1. **Manual Disaster Recovery Documentation**

The process owner will create manual documentation for any automated process which can be used in a disaster recovery scenario where the automated process needs to be done manually.

1. **Front End Web Site**

A standard template web site will be used to collect data and trigger runbooks. This will be written in ASP.Net and leverage a master page. This could be replaced by Service Manager or another application portal site.

1. **Data Input validation**

Data input validation should be done as close to the user as possible. If a front end web page is used to collect data and then trigger a runbook, the web site should validate as much of the data first before triggering a runbook.

1. **Project Request Interview Process**

Project requests for new runbooks should complete an interview process to determine the detailed steps that need to be completed prior to creating the runbook within the Runbook Designer.

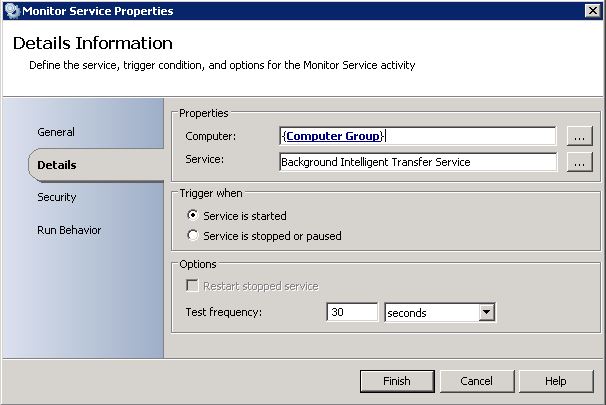
1. **Templates**

Templates should be used for re-usable runbooks. These templates should not be directly linked to but rather copied to the new runbook so that each runbook is self-contained. This ensures that a change to a central runbook template does not impact other runbooks unintentionally.

Custom integration packs should not be directly updated but rather updated custom integration packs should be complied as a new release. Existing workflows should then be updated by using the new actions from the updated integration pack.

1. **Computer Groups**

Typically computer groups should not be used within the Runbook Designer. Groups should be passed to the runbook as a parameter or referenced directly within the runbook itself and the members should be queried dynamically from within the runbook. One exception to this is the use of groups with Monitor activities. Using groups allows you to monitor more than one computer.



1. **Schedules**

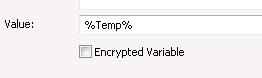
Schedules can be leveraged within the Global Settings section of the Runbook Designer console. Orchestrator will execute schedules used in conjunction with the Check Schedule action. Do not use the Schedule option within the General tab of the runbook properties.

1. **Embedded Credentials**

Ideally the Orchestrator process domain account should be leveraged when accessing network resources. This negates the need for any type of embedded credentials.

If embedded username and password credentials are needed then ensure the password is encrypted by using one of the following methods:

* 1. **Use the encryption option within a variable under Global Settings.**

****

* 1. **Create an integration pack using the Orchestrator Command-Line Activity Wizard with a variable of type Encrypted Text**
  2. **Use the Powershell Integration Pack from Microsoft which allows credentials to be stored securely.**

If passing credentials directly to a runbook, ensure that logging is disabled otherwise the username and password will be available in the log files when the contents of the databus are written to disk.

1. **Operator Web Console**

Operators should be given read only rights to the console so that they can use it for reporting and troubleshooting. They should not have rights to start runbooks from the operator console directly for any runbooks that require input data unless the data is validated during the runbook execution.

1. **Communication**

Orchestrator should leverage tools like Microsoft System Center Configuration Manager to talk directly to the client whenever possible. This ensures that scenarios where computers are offline or at remote locations are properly serviced.

1. **Error Handling**

Runbooks will be designed to notify the process owner of any critical failures. This may be through an email, ticket or simply logging the failure to be reviewed at a future time. Emails that should not receive a reply should utility the sender address of donotreply@<company>.com.

Runbooks must be designed so that lower level errors are not masked or hidden but rather are “bubbled” up to the calling runbook to be processed appropriately.

Utilize Run .Net Script using PowerShell to call the Throw() method to generate custom error messages within your runbook even when logging is disabled.

1. **Validation**

Runbooks must be designed to ensure that the expected work is completed correctly.

Runbooks must be designed to ensure that they do not loop out of control. For example, if an unexpected condition is not handled, the runbook must not be able to do things like generate thousands of emails or tickets or be able to delete every account within active directory or every package within a package share, etc.

1. **Robocopy.exe /MT switch**

Be aware that the /MT switch with robocopy.exe will allow the file copy to be multi-threaded which greatly improved performance however each file that is copied is loaded into memory so be aware of how many instances of robocopy.exe will be allowed to run and the size of files that will be copied. The recommendation is to use /MT:8 by default.

1. **Avoid Long Running Runbooks**

Avoid having non-monitor runbooks executing for more than a day. For example, having a runbook that starts on a Monday and then remains executing and waiting until Friday to complete its task. It is better to store the runbook status in a database and then query it once a day and then on Friday start the runbook again to complete its operation. This will help ensure you can transition runbooks between runbook servers to be able to do maintenance on a runbook server and avoid having the runbook fail from an unexpected mid-week event like a reboot of the server or database, etc.

1. **Use the runbook sanitizer tool to scrub all runbooks exported and imported between environments.**

<http://opalis.wordpress.com/2012/05/12/runbook-sanitation-utility/>

1. **Use the Orchestrator Visio and Word Generator to document your runbook**

The Orchestrator Visio and Word Generator should be used when you receive a request to document how a runbook functions.

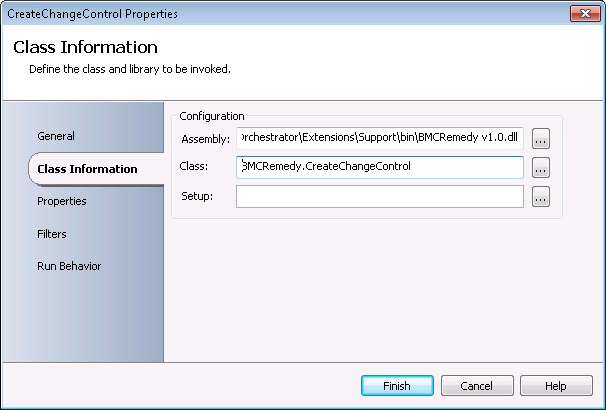
<http://orchestrator.codeplex.com/releases/view/75824>

1. **Version .dll and powershell files created by the Orchestrator Command-Line Activity Wizard.**

This ensures that existing runbooks in production are not updated to use the new integration pack files. This becomes important as large numbers of critical runbooks leverage a custom created integration pack. You can’t always be 100% confident that all combinations of testing have been done on the new integration pack and therefore there is risk in doing a system wide change. By versioning the files, new runbooks can take advantage of the new features and bug fixes in the IP and existing runbooks can be upgraded to use the new IP the next time the runbook is updated for any reason. This allows for regression testing to be done on the runbook after the new IP has been referenced.

1. **Limit the amount of data passed between activities and runbooks.**

<http://blogs.technet.com/b/orchestrator/archive/2012/05/08/orchestrator-quick-tip-what-s-the-maximum-size-of-parameters.aspx>



**Web Service**

When connecting to the Orchestrator Web Service to execute a runbook, avoid using GUID keys to start a runbook. Instead leverage the runbook name. The reason for this is that when importing runbook .oip files the guids for runbooks are modified so in a disaster recovery scenario where the server is rebuilt any existing calls to runbooks would need to be modified. In addition, to support a fail over scenario with a load balancer like F5 where you want to be able to migrate between two live sites the runbooks between these sites will have different guids.

Setup a DNS alias for your production Orchestrator web server. Ensure that all executions of runbooks using the web service reference the DNS alias rather than the server name itself. This will allow you to change the DNS alias to point to another server in the event of a server upgrade or disaster recovery scenario and also positions you to be able to lever a network load balance for high availability.

**Glossary:**

**Runbook** – a process automation flow that automates a defined set of tasks. Example: “Import XML File”

**Workflow** – A collection of linked runbooks that automate a larger set of tasks. Example: “Create New Software Delivery Package Definition”

**Project** – A collection of related workflows used to automate a group of processes. Example: “Software Package Management v1.0”

**Microsoft Orchestrator** – The software application from Microsoft that is used to generate and execution runbooks.

**Appendix:**

### Orchestrator Data Manipulation Functions

|  |  |  |
| --- | --- | --- |
| Name | Description | Example |
| Field | String function used to return specific text from a delimited list (e.g. split function). | [Field(‘Field1;Field2;FieldN’,’;’,2)]Returns ‘Field2’ |
| Sum | Arithmetic function used to return the sum of a set of numbers. Can also be used to promote a number with a string format to integer format. | [Sum(1,10,100,1000)]Returns 1111 |
| Mid | String function used to return specific subset of text from a superset string of text. | [Mid(‘Return subset from this string’,8,6)]Returns ‘subset’ |
| Instr | String function used to return the position of the first occurrence of specific text from a superset string of text. | [Instr(‘Find me’,’me’)]Returns 6 |
| Len | String function used to return the length of a specific text string. | [Len(‘four’)]Returns 4 |
| Upper | String function used to return text after it has been converted to upper case. | [Upper(‘upper me’)]Returns ‘UPPER ME’ |
| Lower | String function used to return text after it has been converted to lower case. | [Lower(‘LOWER ME’)]Returns ‘lower me’ |
| Trim | String function used to return text after removing leading and trailing spaces. | [Trim(‘ Trim me ‘)]Returns ‘Trim me’ |
| Rtrim | String function used to return text after removing trailing spaces. | [Rtrim(‘ Trim me ‘)]Returns ‘ Trim me’ |
| Ltrim | String function used to return text after removing leading spaces. | [Ltrim(‘ Trim me ‘)]Returns ‘Trim me ’ |
| Left | String function used to return text of specified length from the left side of a superset string of text. | [Left(‘Left to Right’,4)]Returns ‘Left’ |
| Right | String function used to return text of specified length from the right side of a superset string of text. | [Right(‘Left to Right’,5)]Returns ‘Right’ |
| Diff | Arithmetic function used to return the difference of two numbers. The 3rd parameter is for precision which is option but allows you to specify how many decimal places to return. | [Diff(100.23,95.65,2)]Returns 4.58 |
| Div | Arithmetic function used to return the quotient of two numbers. | [Div(22,7,2)]Returns 3.14 |
| Mult | Arithmetic function used to return the product of a set of numbers) | [Mult(8,16,32)]Returns 4096 |

### \*\*Note that nesting data manipulation functions is allowed.

### Example: [Field(Field(Field(‘Parse the;TEXT#MSG:out of this;string’,’;’,2),’:’,1),’#’,1)]

### Returns ‘TEXT’

When using the data manipulation functions and the function is not being calculated, try replacing the single quote characters. There have been issues with invalid single quote characters coming from copy and paste commands.