Table of contents

Get WFC::Factory up & running

Prerequisites

To use WFC::Factory you need:

- Powershell Core installed on either **Linux**, **Mac** or **Windows**.
- Optional: WFC::REST when you want to import your created solutions
- Optional: PSGraph module & GraphViz installation if you want to preview workflow dependencies

Notice: you can get the multiplatform Powershell Core here: Powershell GitHub or use your favorite Package Manager (yum, apt, chocolatey etc.).

Installation

Copy the WFC::Factory files (or clone repo) into a directory that is recorded in $env:PSModulePath and:

```
Import-Module WFC-Rest-Factory
```

... or store the directory whereever you want and import the module using full directory information.

```
Import-Module 'c:\data\WFC-REST-Factory'
```

Linux & Mac

If you intend to run WFC::Factory on a Unix-based platform, you must set the below variables after importing the module.

```
# This defines where the configuration will be stored/loaded from by default
$env:APPDATA = '/home/myuser'
# If you run the Pester tests, please provide a temporary directory
$env:tmp = '/tmp'
```

Superduper - quickstart

Pester is a test framework for Powershell. Each an every Cmdlet in WFC::Factory has it's Pester tests that can be executed at any time. WFC::Factory tests do not run against your Automation Engine (as known from WFC::REST tests).

```
# This will run the tests against your environment. If you get an error here,
# you might need to install pester first (install-module pester)
Invoke-Pester
```

Why should I run the tests?

By running the test, your Powershell setup will be tested. There might be differences or incompatibilities with your Powershell Core environment compared with my dev environment. The tests try to determine, whether the Cmdlets would run properly on your system with no effort from your side.

Available Cmdlets

Generic Objects

New-AutomicObject

The New-AutomicObject loads a JSON object definition file into memory for further processing using the WFC::REST Factory Cmdlets. You might place additional .json object "templates" in the data subdirectory of the WFC::REST Factory module (exports can be done using the WFC::REST Cmdlet Export-AutomicObject).

Note: The object_type parameter supports autocompletion to show the available object definitions.

```
# Use existing workflow as input
$jobp = New-AutomicObject -object_name JP.DEMO -location /TARGET_PATH -object_type
jobp_standard
```

Workflow Management

The below Cmdlets can be executed on an empty or preexisting workflow. To create a new workflow you can use the New-AutomicObject cmdlet. To load an existing workflow, you might use WFC::REST first to gather the JSON definition.

```
# Use existing workflow as input
$jobp = Export-AutomicObject -object_name 'JP_DEMO' -out

# Create workflow from scratch (JSON templates are in data subfolder)
$jobp = New-AutomicObject -object_name 'JP_DEMO' -object_type 'JOBP_STANDARD' -
location '/DEMO_OBJECTS'
```

Get-AutomicJobpTask

Extract the task definition from a loaded workflow.

```
# Returns 0,1 result
$reference = Get-AutomicJobpTask -jobp $jobp -lnr 1

# Returns 0,n result
$reference = Get-AutomicJobpTask -jobp $jobp -object_name 'JOBF.DEMO'

# Returns 0,n result
$reference = Get-AutomicJobpTask -jobp $jobp -object_type 'JOBF'
```

New-AutomicJobpTask

Create a new task in a jobp. Returns the complete task definition, however changes to the $reference object will not be updated in the jobp! The task reference is important because it contains the unique line_number identifier within the jobp object.

You might also pass on the full task definition.

```
$reference = New-AutomicJobpTask -jobp $jobp -object_name 'JOBS.WIN.EXPORT_DATA1'
-object_type 'JOBS'

# Add task by a previous reference. line_number of $another_reference will differ
from $reference.
$another_reference = New-AutomicJobpTask -jobp $jobp -task $reference

# Copy & paste task. Also here $another_task will have a different line_number
value.
$task = Get-AutomicJobpTask -jobp $jobp -object_name 'JOBF.DEMO'
$another_task = New-AutomicJobpTask -jobp $jobp -task $task
```

Remove-AutomicJobpTask

Remove task from a workflow. Edges will also removed.

```
# Remove by lnr
Remove-AutomicJobpTask -jobp $jobp -task 2

# Remove by reference
$task = Get-AutomicJobpTask -jobp $jobp -object_name 'JOBF.DEMO'
Remove-AutomicJobpTask -jobp $jobp -task $task
```

Connect-AutomicJobpTask

Connect two tasks with each other.

```
Connect-AutomicJobpTask -jobp $jobp -task 2 -dependsOnTask 1 -ok_status 'ANY_OK'

# Create two tasks and connect them
$export = New-AutomicJobpTask -jobp $jobp -object_name 'JOBS.EXPORT'
$copy   = New-AutomicJobpTask -jobp $jobp -object_name 'JOBF.COPY'
Connect-AutomicJobpTask -jobp $jobp -task $copy -dependsOnTask $export -ok_status
'ANY_OK'
```

Disconnect-AutomicJobpTask

Removes the connection between two tasks.

```
Connect-AutomicJobpTask -jobp $jobp -task 2 -dependsOnTask 1 -ok_status 'ANY_OK'

# Create two tasks and connect them
$export = New-AutomicJobpTask -jobp $jobp -object_name 'JOBS.EXPORT'
$copy   = New-AutomicJobpTask -jobp $jobp -object_name 'JOBF.COPY'
Connect-AutomicJobpTask -jobp $jobp -task $copy -dependsOnTask $export -ok_status
'ANY_OK'
```

Get-AutomicJobpTaskConnection

Get / check a connection between two tasks. Can also identify all dependencies / successors of a task.

```
# Returns all predecessor - dependencies of task with line_number 2
Get-AutomicJobpTaskConnection -jobp $jobp -task 2

# Returns all successor - dependencies on the task "JOBF.COPY_DATA"
$task = Get-AutomicJobpTask -jobp $jobp -object_name 'JOBF.COPY_DATA'
Get-AutomicJobpTaskConnection -jobp $jobp -dependsOnTask $task

# Returns the connection between task 1 and 2 - if there is any.
Get-AutomicJobpTaskConnection -jobp $jobp -task 2 -dependsOnTask 1
```

Connect-AutomicJobpOpenEndTask

Automatically connects open ends with either or depending on whether the "open-end" task has missing successors or predecessors.

```
Connect-AutomicJobpOpenEndTask -jobp $jobp
```

Repair-AutomicJobpGap

This Cmdlet should be run once the complete jobp has been built to ensure, that the line_numbers of the tasks are incrementing properly. There might be gaps between the line_numbers if tasks have been removed during the build process.

```
Repair-AutomicJobpGap -jobp $jobp
```

Get-AutomicJobpPath

Returns an array of arrays, showing the possible paths through the worklow. Tasks are referred by their unique line_number, array is sorted from longest path to shortest path already.

This is a feature used for automatic coordinate detection. In case you'll find a better algorithm than JLPF (Joels Longest Path First), feel free to implement a reference algorithm and let me know 😁.

```
Get-AutomicJobpPath -jobp $jobp
```

Set-AutomicJobpTaskCoordinate

This Cmdlet will assign proper X/Y coordinates (row & columns) to all tasks. It is necessary to run this Cmdlet if you add new tasks (New-AutomicJobpTask) without specifying -column & -row.

Be warned, this cmdlet will ignore existing coordinates and assign new coordinates on it's own. Your workflow layout might change but logic will remain.

```
Set-AutomicJobpTaskCoordinate -jobp $jobp
```

Build-AutomicJobp

This is a enclosing cmdlet that links open ends, assigns task coordinates and makes sure, that there are no line_number gaps. You should execute this Cmdlet before importing it to the AE after you added or removed tasks on a workflow. This might shuffle your task locations, however so your workflow might look different afterwards.

```
Build-AutomicJobp -jobp $jobp
```

Show-AutomicJobp

Preview a workflow JSON. This will convert the workflow to a GraphViz visualization.

You must have the Powershell Module psgraph installed as well as GraphViz.

```
# Make sure you have installed the PSGraph module first
# Install-Module psgraph
Show-AutomicJobp -jobp $jobp
```

Get-TaskLnr

Returns the line_number element of a task or the integer value if a integer value has been specified.

```
$lnr = Get-TaskLnr -task $task
```

Repair-AutomicJobpTaskConnection

If a task has multiple predecessors, the connections have an iterating identifier that must start at 1 and increment for each predecessor. If tasks have been removed or new connections added / removed, this Cmdlet will fix the numbers.

```
Repair-AutomicJobpTaskConnection -jobp $jobp
```

Set-AutomicJobpTaskCondition

Set the task pre- / post-condition. It is strongly recommended, to "copy" the preconditions from a reference task first using Get-AutomicJobpTaskCondition since the data structure is complex.

```
# Simple copy from one task and paste in other task. This considers
# pre and post conditions
$conditionTask = Get-AutomicJobpTask -jobp $jobp -object_name 'JOB.SOURCE'
$cond = Get-AutomicJobpTaskCondition -jobp $jobp -task $conditionTask

$targetTask = Get-AutomicJobpTask -jobp $jobp -object_name 'JOB.TARGET'
Set-AutomicJobpTaskCondition -jobp $jobp -task $targetTask -condition $cond

# If target task has conditions set already, you might need to -force the pate
Set-AutomicJobpTaskCondition -jobp $jobp -task $targetTask -condition $cond -force
```

Get-AutomicJobpTaskCondition

Gather the conditions of a task. You must first use Get-AutomicJobpTask to get the task reference (or specify lnr directly if known).

```
$conditionTask = Get-AutomicJobpTask -jobp $jobp -object_name 'JOB.SOURCE'
$cond = Get-AutomicJobpTaskCondition -jobp $jobp -task $conditionTask
```

Remove-AutomicJobpTaskCondition

Removes the conditions of a task.

```
$noCondsTask = Get-AutomicJobpTask -jobp $jobp -object_name 'JOB.NOCONDS'
$cond = Remove-AutomicJobpTaskCondition -jobp $jobp -task $noCondsTask
```

Set-AutomicJobpTaskGeneric

Set generic task settings (active/inactive, breakpoint, alias name).

```
$newTask  = New-AutomicJobpTask -jobp $jobp -object_name "JOBS.ONE" -object_type
"SCRI"
Set-AutomicJobpTaskGeneric -jobp $jobp -task $newTask -alias "INACTIVE_TASK" -
active $false
```

Set-AutomicJobpTaskCalendar

Set a tasks calendar conditions. Conditions must be passed as array of hashtables @(@{ cale = key }, @{ cale = key }, ...)

```
$newTask  = New-AutomicJobpTask -jobp $jobp -object_name "JOBS.ONE" -object_type
"SCRI"
Set-AutomicJobpTaskCalendar -jobp $jobp -task $newTask -conditionMatch
allConditionsMatch -calendarCondition @( @{ 'UC_HOLIDAYS.A' = 'CHRISTMAS'} )
```

Set-AutomicJobpTaskVariable

Set the object or promptset variables of a task.

```
# Set object variable &OBJ_VAR# to "Hello world" for task with lnr 3
Set-AutomicJobpTaskVariable -jobp $jobp -task 3 -variableName 'OBJ_VAR#' -value
'Hello world'

# To set a promptset value is more complex, since there can be the same variable
name in multiple promptsets.
Set-AutomicJobpTaskVariable -jobp $jobp -task 3 -promptset 'PRPT.PROMPTSET_NAME' -
variableName 'OBJ_VAR#' -value 'Hello world'
```

Remove-AutomicJobpTaskVariable

Removes the object or promptset variables of a task. This Cmdlet can be extremely useful if you want to remove variables that have been accidentially set on workflow level.

```
# This removes all task-level object & promptset variables
Remove-AutomicJobpTaskVariable -jobp $jobp

# This removes all task-level object & promptset variables if they are named
"TASK_LVL#"
Remove-AutomicJobpTaskVariable -jobp $jobp -variableName 'TASK_LVL#'
```