Table of contents

Get WFC::REST up & running

Prerequisites

To use WFC::REST you need:

- "Windows Powershell" OR "Powershell Core" installed on either **Linux**, **Mac** or **Windows**.
- An user on your Automic system you can also login with.
- URL of the REST endpoint.
- Network access to the REST endpoint (no blocking firewalls).

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

Installation

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

```
Import-Module WFC-REST
```

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

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

Linux & Mac

If you intend to run WFC::REST 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::REST has it's Pester tests that can be executed at any time. The idea is, that your AE system won't be harmed by the tests. Anyway, objects might be created, overwritten, deleted or started. So you *must* register a Pester endpoint & credential that is *either on a* *dedicated client* or at least *a non-critical dev / test system*. Once this precondition is met, you can execute the tests against your AE system with the below commands:

```
# Register identified endpoints
New-AutomicEndpoint -endpointAlias 'wfc_rest_pester_endpoint' -endpointUrl
'https://ae.intra.ch:8080/ae/v1/9000'
New-AutomicCredential -credentialAlias 'wfc_rest_pester_credentials' -interactive

# 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?

Two reasons:

- Pester will run various checks on the module. Your infrastructure and compatibility will be tested. If you run into an issue, you can ask for community assistance or report it and get it fixed.
- Pester Test Cases are a good source of examples on what's possible with the WFC-REST module. You can find tests on all Cmdlets below on the "tests" folder.

Create endpoint & credential configuration

The module must have a clue on where to connect & what credentials to use. This section is about making this information available to WFC::REST.

(De-)Register Endpoint Alias

Register a REST URL with an Alias.

```
# Register Endpoint with alias PROD pointing to client 9000.
New-AutomicEndpoint -endpointAlias 'PROD' -endpointUrl
'https://ae.intra.ch:8080/ae/v1/9000'

# Force-Override existing PROD alias
New-AutomicEndpoint -endpointAlias 'PROD' -endpointUrl
'https://ae.extra.ch:8080/ae/v1/9000' -override

# Remove endpoint alias.
Remove-AutomicEndpoint -endpointAlias 'PROD'
```

(De-)Register Credentials Alias

Maps credentials to a REST endpoint for later use.

```
# Register Credentials with alias personalUser. A prompt will pop up asking for
username & password.
New-AutomicCredential -credentialAlias 'personalUser' -interactive

# Provide username & passwort via commandline
New-AutomicCredential -credentialAlias 'personalUser' -userName 'joel' -password
'pwd'

# Provide existing credentials object
$cred = Get-Credential
New-AutomicCredential -credentialAlias 'personalUser' -cred $cred

# Remove endpoint alias.
Remove-AutomicCredential -credentialAlias 'personalUser'
```

Store configuration

Exports all credentials & endpoint aliases. Credentials are securely exported.

```
# To store the currently loaded configuration to the default location
Export-WFCRestConfiguration

# Specify file to store configuration into
Export-WFCRestConfiguration -file 'c:\config\test.json'
```

Load configuration

Imports a previously exported configuration.

The credentials will only be decrypteable with the same user on the same host. Copying the file to another host or share it with another user on the same system won't work.

```
# To load the default configuration
Import-WFCRestConfiguration

# Specify file to load configuration from
Import-WFCRestConfiguration -file 'c:\config\test.json'
```

Show available aliases & credentials

Show what endpoints and credential aliases are available.

```
Get-WFCRestConfiguration
```

Set active connection

Set the default connection. If you use WFC::Rest Cmdlets without specifying a connection, this will be used. Also this will change your command prompt.

```
Set-AutomicConnection -endpointAlias 'PROD' -credentialAlias 'personalUser'
```

Clear active connection

Reset default connection.

```
Clear-AutomicConnection
```

User settings

Usermode

WFC::REST provides you two methods to work with the Automic REST interface. One is programming-focused while the other method is enduser-focused. The difference is, that in the programming-mode, all Cmdlets will return the plain REST answer while the enduser mode will represent the data in a way that suits the user.

```
# Activate the user mode
Set-WfcRestSetting -usermode $true
```

You will notice, that the prompt will slightly change (if you used Set-AutomicConnection). The programming mode is indicated by curly brackets around your connection:

```
{default@test} PS C:\
```

The enduser mode is indicated by square brackets:

```
[default@test] PS C:\
```

The impact of this setting is imense. If you search objects in programming-mode, the result might look like this:

```
data
----
{@{name=JP_DEMO; type=JOBP; sub_type=; platform=; id=1000428; title=;
```

```
archive_key1=; archive_key2=;...
```

While in usermode, the representation is userfriendly:

```
Name                 Type  Title  Location
----                 ----  -----  --------
JP_DEMO              JOBP         \TEST
```

In usermode you still have all the information, it's just not displayed to the user.

Available Cmdlets to work with the AE

All examples below will only work, if you used Set-AutomicConnection before.

```
# With Set-AutomicConnection
Set-AutomicConnection -endpointAlias 'DEV' -credentialAlias 'personalUser'
Start-AutomicObject -object_name 'DEMO'
Start-AutomicObject -object_name 'DEMO2'

# Without Set-AutomicConnection, you must specify the endpoint & credential alias
to use
Start-AutomicObject -object_name 'DEMO' -endpointAlias 'DEV' -credentialAlias
'personalUser'
Start-AutomicObject -object_name 'DEMO2' -endpointAlias 'DEV' -credentialAlias
'personalUser'
```

Most Cmdlets below feature more than the parameters shown in the examples. If you feel that an important functionality has not been featured in the examples, let me know.

Runtime changes (Start, Stop, Restart, Cancel...)

Start-AutomicJob

Starts an object & returns it's RunID.

```
# Execute immediately
Start-AutomicJob -object_name 'OBJECT#JOBS'

# Execute immediately and pass variable input
Start-AutomicJob -object_name 'OBJECT#JOBS' -variables @{
   'NAME#'     = 'Joel'
   'SURNAME#'  = 'Wiesmann'
}
```

```
# Once-start the object ONCE#JOBS in 24h
Start-AutomicJob -object_name 'ONCE#JOBS' -executeAt ([datetime]::now.AddDays(1))

# Start Object PERIOD#JOBS every Wednesday and Friday on 9AM
Start-AutomicJob -object_name 'PERIOD#JOBS' -start_at 09:00 -execute_weekly
wed,fri

# Start object with WEEKDAY event from CALE_ALL every minute
Start-AutomicJob -execute_calendar_list @{ "calendar" = 'CALE_ALL'; "event" =
"WEEKDAY" } -object_name 'WFC_REST_PESTER_SCRI' -execute_interval 'P0DT0H1M'

# ... and much more. This Cmdlet features many parameters. Feel free to explore.
```

Restart-AutomicJob

Restarts a previous job run.

Warning - if your object has a promptset, you must define all variables explicitly.

```
# Restart a job
Restart-AutomicJob -run_id 123456

# Restart a job and overrule the promptset values
Restart-AutomicJob -run_id 123456 -variables @{ 'INPUT_FIELD#' = 'Demo Value' }
```

Cancel-AutomicJob

Cancels a running job.

```
# Cancel a job
Cancel-AutomicJob -run_id 123456

# Cancel a jobp recursively
Cancel-AutomicJob -run_id 123456 -recurse
```

Start-AutomicScript

CallAPI solution via REST. Starts AE Script directly.

```
# One-line solution
Start-AutomicScript -script ':PRINT "Hello World"'

# Multi-line solution
$script = @(
        ':PRINT "Hello world"',
        ':PRINT "Another line"'
)
```

```
Start-AutomicScript -script $script
```

Add-AutomicJobComment

Adds a comment to a running execution.

```
# Simple as that
Add-AutomicJobComment -run_id 123456 -comment 'Hello. It is me.'
```

Object export & import

Export-AutomicObject

Exports an object into the new JSON format.

```
# Exports the object into the current working path with the name <object>.json
Export-AutomicObject -object_name 'WFC_REST_PESTER_SCRI'

# Export of the object into a specific file
Export-AutomicObject -object_name 'WFC_REST_PESTER_SCRI' -file
'D:\pester_file_export.json'

# Export of the object into a specific path (name of json is <object>.json)
Export-AutomicObject -object_name 'WFC_REST_PESTER_SCRI' -folder $env:tmp

# Export of the object into a variable without file
$data = Export-AutomicObject -object_name 'WFC_REST_PESTER_SCRI' -out
```

Import-AutomicObject

Imports a JSON file or object variable (i.e. coming from a previous Export-AutomicObject).

```
# Import JSON file to /DATA
Import-AutomicObject -file (Join-Path $env:tmp 'pester_scri.json') -path '/DATA'

# Import [psobject] variable (from Export-AutomicObject) to /DATA
Import-AutomicObject -data $scriObject -path '/DATA'

# Overwrites target object (not enabled by default)
Import-AutomicObject -data $data -path '/DATA' -overwrite
```

Job information Cmdlets

Get-AutomicJob

Read the execution details of a runid (i.E. status).

```
# Get execution details of RunID 123456
Get-AutomicJob -runId 123456

# Get execution details and information about the available report types &
comments
Get-AutomicJob -runId 123456 -fields 'reports','comments'
```

Get-AutomicVariable

Returns the published variables of a RunID (similiar to Get-AutomicJob -fields 'variables').

```
Get-AutomicJobVariables -runId 123456
```

Get-AutomicJobERT

Returns the estimated runtime of a jobplan. If the run-id of a non-jobp object or a ENDED_* runid is specified, this will throw an exception.

```
Get-AutomicJobERT -runId 123456
```

Get-AutomicJobChildren

Get the childs of a jobplan. If the run-id of a non-jobp object is specified, this will return no results (activate_uc_object activations are being ignored).

```
Get-AutomicJobChildren -runId 123456
```

Get-AutomicJobReportType

Returns the available job report types of a runid (ACT, REP, etc). Similiar to Get-AutomicJob -fields 'reports'.

```
Get-AutomicJobReportType -runId 123456
```

Get-AutomicJobReport

Reads the report(s) of a runid.

```
# Get the activation report of RunID 123456
Get-AutomicJobReport -runId 123456 -reportType ACT
```

```
# Get the common report (REP) of RunID 123456
Get-AutomicJobReport -runId 123456

# Get max. three pages instead of one
Get-AutomicJobReport -runId 123456 -max_results 3
```

Get-AutomicJobComment

Read comments of a runid. Similiar to Get-AutomicJob -fields 'comments'.

```
# Get the comments of runId 123456
Get-AutomicJobComment -runId 123456
```

Get-AutomicActivity

Returns the activities.

```
# Return all activities
Get-AutomicActivity

# Return activities of given object name
Get-AutomicActivity -name 'DEMO#JOBS'

# Filter activities by object type & name
Get-AutomicActivity -type 'JOBS','JOBF','JOBP' -name 'DEMO#*'
```

Get-AutomicObjectInput

Returns the available input fields of a Job if one or multiple promptsets have been attached to it.

```
# Get the attached promptsets and their input fields
Get-AutomicObjectInput -object_name 'DEMO_PRPT_OBJECT'
```

Find-AutomicObject

Searches for an Automic object.

WARNING #1: Search logic on object names is fuzzy!

WARNING #2: Search is not realtime updated after creation / modification of objects!

```
# Common object name search
Find-AutomicObject -object_name 'DEMO*'
```

```
# Search using AE object path non-recursive & recursive
Find-AutomicObject -location '/PESTER'
Find-AutomicObject -location '/PESTER' -recursive

# Search for VARA having a record named 'DEMO_KEY' and ArchiveKey1 (or 2) equals
'demo key'
Find-AutomicObject -variable_key 'DEMO_KEY' -archive_key 'demo key'

# Search for SCRI having a process :PRINT "WHY"
Find-AutomicObject -process ':PRINT "WHY"' -object_type SCRI

# ... and much more. This Cmdlet features so many parameters, it's a pain ;-)
```

System information & management

Get-AutomicClient

Get the available clients of the system including some extra information like amount of objects and if client is online or not. If you do not run this command on a client 0 connection, only the currently logged in client is shown.

```
Get-AutomicClient
```

Remove-AutomicClient

Removes a client from the system. This cmdlet only works if you are running on a client 0 connection.

```
Remove-AutomicClient -client 1234
```

Test-AutomicClient

Check if the REST interface is online.

```
Test-AutomicClient
```

Get-AutomicHealth

Get system status information.

```
Get-AutomicHealth
```

Get-AutomicFeatures

Returns the available features. This seems to be a base query endpoint for upcoming plugins.

```
Get-AutomicFeatures
```

Useful helpers

Watch-AutomicJob

Wait for a job to end.

Timeouts specified here are not timeouts on the AE object but timeout on the Cmdlet to wait for the job to end.

```
# Start Object DEMO#JOBS and wait for it to end (polling 5 seconds, no timeout)
Start-AutomicJob -object_name 'DEMO#JOBS' | Watch-AutomicJob

# Start Object DEMO#JOBS and wait for it to end with a polling interval of 60 seconds (long running jobs)
# and a timeout of 1h
Start-AutomicJob -object_name 'DEMO#JOBS' | Watch-AutomicJob -interval 60 -timeout_sec 3600

# Start Object MULTIDAY#JOBS and timeout in three days at 11am
Start-AutomicJob -object_name 'MULTIDAY#JOBS' | Watch-AutomicJob -interval 60 -timeout_datetime ([datetime]::now).Date.AddDays(3).addHours(11)
```