# SW ETL: Python Script for automating Scheduler process execution

The following describes the process and requirements for a python application to facilitate SW's data-automation needs with the new cloud-model.

# Purpose:

The purpose of this script is to facilitate ETL automation needs that cannot be met by out-of-the-box functionality of the client's cloud software. This script will enable the client to execute and monitor process execution and completion with more timely-precision and report on process-execution outcomes.

# Overview:

The client uses an enterprise process-automation (BMC) software that extracts data from oracle databases into flat-files, submits the files to cloud-sFTP servers, and executes accompanied data-imports in another database. This process involves ~300 data-files.

The cloud-software does not provide flexibility for the client to execute these data-imports with timely-precision or report back to the BMC application whether the imports were successful or not. This is required as there are dependencies.

For this reason, this python application will be developed to execute these data-imports with an API call. Additionally, the script will monitor if the accompanied data-files have been successfully transmitted to the sFTP server (this is achieved by the presence of a trigger-file. If the trigger-file is present, it signifies that the data-file has been submitted successfully), monitor process execution (data-import cannot be executed if another global-action is running, and monitor the process until the import concludes), and report output an variable (success or failure) back to the BMC application.

# Assumptions:

* Python 3.4 or higher will be used for the development of this script
* SW will prepare a Windows environment to store and execute this script
* SW will configure their BMC application to execute this script and pass 3 variables
  + Variable1: a string indicating the scheduler-process-name
  + Variable2: a Boolean indicating whether a check for trigger-files is required
  + Variable3: a list of trigger-files (optional variable, only Variable2 = true)

# Requirement Overview:

## [Part-1] Capture input variables

### 1.1 parse and capture all input-variables

* client will execute this script from command-prompt and we expect two mandatory variables and 1 optional variable (see above for types)
* how can these variables be passed into the script?
* How can these be parsed/captured?
* Terminate script if variable1 and variable2 are not provided
* If variable2 = true, variable 3 must be provided (terminate if not provided)

## [Part-2] Check Validity of given Variables

### 2.1 Determine scheduler-process-ID from scheduler-process-name

- client provides process-name (variable#1), but process-execution requires process-ID

- the following API endpoint can be used to obtain process-ID from process-name

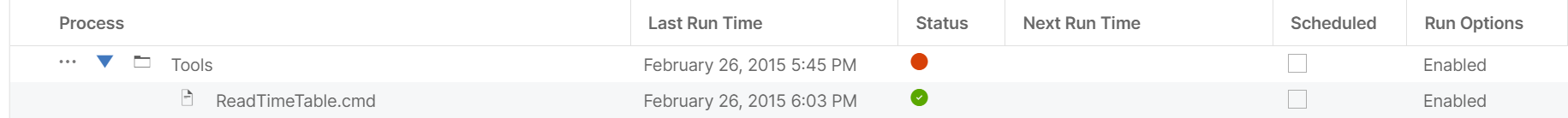
- in the system, there could be 100s of processIDs, each produces an JSON output as shown below.

URL = base\_URL + 'scheduleitem/'

Sample JSON output of API call. Every scheduler-process has a uniqueID:

{'id': 1, 'name': 'Tools', 'scheduleItemType': 'Folder', 'order': 0, 'lastRun': '2015-02-26T22:45:26.49Z', 'lastRunStatus': 'Error', 'activation': 'Enabled', 'nextRun': '0001-01-01T00:00:00', 'childScheduleItems': [{'executable': 'ReadTimeTable.cmd', 'arguments': 'ReadTimeTable.cmd', 'id': 10, 'name': 'ReadTimeTable.cmd', 'scheduleItemType': 'ExternalTool', 'order': 0, 'lastRun': '2015-02-26T23:03:44.78Z', 'lastRunStatus': 'Success', 'activation': 'Enabled', 'nextRun': '0001-01-01T00:00:00', 'parent': 1, 'settings': {'schedulerSettingsId': 0, 'scheduleItemId': 10, 'emailOnFailure': False, 'emailOnSuccess': False, 'stopOnFailure': False, 'stopToolOnTimeout': False, 'isGlobal': True, 'overrideChildSettings': False, 'successEmails': [], 'failEmails': [], 'externalToolTimeout': 0, 'enableRetries': False, 'version': {'rowVersion': 90182}}, 'version': {'rowVersion': 94037}}], 'settings': {'schedulerSettingsId': 0, 'scheduleItemId': 1, 'emailOnFailure': False, 'emailOnSuccess': False, 'stopOnFailure': False, 'stopToolOnTimeout': False, 'isGlobal': True, 'overrideChildSettings': False, 'successEmails': [], 'failEmails': [], 'externalToolTimeout': 0, 'enableRetries': False, 'version': {'rowVersion': 90182}}, 'version': {'rowVersion': 94036}}

Example of the scheduler process in the cloud tool:



### 2.2 Terminate script if scheduler-process-name is not found in cloud-system

If process-name doesn’t exist, terminate the process

### 2.3 Determine availability of trigger files in sFTP with Loop

* if the variable2=true perform this step, otherwise skip to Part-3
* cycle through the list of trigger-files (variable#3) and for each, check for availability of the corresponding file in the cloud-sFTP repo

#API Call for the cloud-sFTP repo which provides a list of the files in the repo.

URL = base\_URL + 'serverfilenames?filter=FileType%3DData%3BIsComplete%3Dtrue '

Sample json output of API call:

['2021-04-09-191546806.csv', 'AccountTest.txt', 'Book1.xlsx', 'Copy of DDD 1.xlsx', 'CurrencyList.xlsx', 'Determine Monthly Credits.xlsx', 'doc1.txt', 'doc2.txt', 'doc3.txt', 'doc4.txt', 'doc5.txt', 'doc6.txt', 'dtSales.xlsx', 'dtTransactions data 1.xlsx', 'dtTransactions data.xlsx', 'PayeeDocumentBatchUpload.zip', 'SUNDY-2021-04-09-191618951.csv', 'symonsales.csv', 'symonsales.xlsx']

Trigger-files may not be ready at time of script execution. Perform the above check for X minutes by looping the API call (Loop every few seconds until X minutes passed). Terminate script if trigger-files are not available after X minutes. Proceed to next step if trigger-files are found.

## [Part-3] Execute the scheduler-process

### 3.1 Monitor the system until global action can be executed

- system may be in the process of running another global-action. During this time, scheduler-processes cannot be executed

- monitor the system until its ready (loop every few seconds until X minutes passed)

- send a GET request call for the global action status. Returns True if a global action is running, which means no other global action can be run at this time. Returns False if no global action is running, which means we are good to proceed and execute the scheduler-process.

URL = base\_URL + 'globalactionstatus'

Sample JSON output of API call:

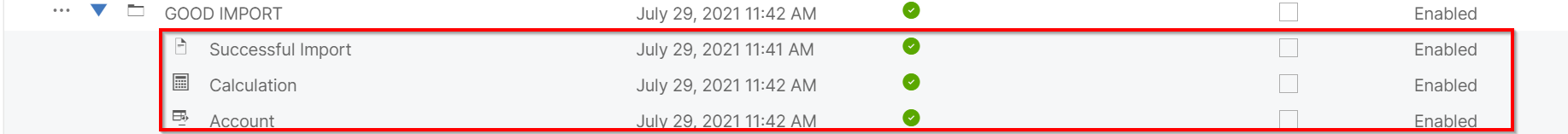
False

### 3.2 Execute the scheduler-process

- to execute the process, send a POST request to the below URL by identifying the scheduler-process-ID (obtained from Part2)

URL = base\_URL + 'rpc/scheduleitem/'+str(process-ID)+'/run'

Use a POST request to run the scheduler process (a folder with many tasks, such as imports, process updates, etc.). Running the scheduler process will run all the child tasks. In the example below, if Variable1 = ‘GOOD IMPORT’, and we use the scheduler-process-ID obtained in step 2.1 to run this process. The following will all run:



Once the scheduler-process executes, the script must monitor its completion. The underlying tasks in any process can fail for multiple reasons. Moreover, execution of said process can vary greatly – maybe 5seconds or 5hours.

In order to monitor this process’ status, we require the system’s instanceID for this process. This can be obtained from the JSON of the above POST call. The systems instanceID is shaded in yellow:

JSON output:

{'completedactivities': 'api/v1/completedactivities/11486', 'liveactivities': 'api/v1/liveactivities/11486'}

To monitor the status of the process, send a GET request to the following URL

URL = base\_URL + 'liveactivities/'+str(instanceID)

The status\_code of the above API call is either 200 or 404

200 means the process is still running, i.e. it’s active

404 means the process is complete

Perform the above check using a loop every few seconds until the process completes.

Example:

Status\_code = 200

JSON = {'progressId': 11486, 'userId': 'ivomid', 'type': 'Scheduler', 'status': 'Running', 'time': '2021-07-31T17:06:17.32Z', 'apiServer': 'sjc03-prd-api07', 'percent': 0, 'hasDescription': False, 'expiresAt': '2021-07-31T17:15:17.32Z', 'isCancellable': True, 'isInitialization': False, 'computationId': -1}

Or

Status\_code = 404

JSON = {'Message': 'Live activity with ID 11486 was not found.'}

Once the process completes, stop the loop and proceed to next step

### 3.3 Determine if the scheduler-process completed successfully.

As a final check, we need to report on process-completion status, as well as details

The idea is to report an output variable= 0 for pass or output variable = 1 for failure

To determine pass or failure, we submit a GET request to the following URL. Remember processID was obtained in Part2.1.

URL = base\_URL + 'scheduleitem/'+str(process-ID)

From the below JSON output, capture the green and blue-shaded variables.

Green highlights the process-level information. If lastRunStatus = Error, produce a variable 1 as output. If lastRunStatus = Success, report 0 as output.

Blue highlights are the task-level information. For every task, extract the shaded details for the logs.

JSON output:

{'id': 1020, 'name': 'BAD IMPORT', 'scheduleItemType': 'Folder', 'order': 5, 'lastRun': '2021-07-31T17:06:24.483Z', 'lastRunStatus': 'Error', 'activation': 'Enabled', 'nextRun': '2021-07-31T17:30:00.0003804Z', 'scheduledTime': {'id': 3, 'scheduleItemId': 1020, 'minute': 30, 'hour': 17, 'dayOfMonth': -1, 'month': -1, 'dayOfWeek': -1, 'nextRunTime': '2021-07-31T17:30:00.0003804Z'}, 'childScheduleItems': [{'id': 1021, 'name': 'Bad Import', 'scheduleItemType': 'Import', 'order': 1, 'lastRun': '2021-07-31T17:06:23.277Z', 'lastRunStatus': 'Error', 'activation': 'Enabled', 'nextRun': '0001-01-01T00:00:00', 'parent': 1020, 'childScheduleItems': [], 'settings': {'schedulerSettingsId': 0, 'scheduleItemId': 1021, 'emailOnFailure': False, 'emailOnSuccess': False, 'stopOnFailure': False, 'stopToolOnTimeout': False, 'isGlobal': True, 'overrideChildSettings': False, 'successEmails': [], 'failEmails': [], 'externalToolTimeout': 0, 'enableRetries': False, 'version': {'rowVersion': 90182}}, 'version': {'rowVersion': 820549}}, {'id': 1025, 'name': 'Account\_', 'scheduleItemType': 'ClearTable', 'order': 2, 'lastRun': '2021-07-31T17:06:23.607Z', 'lastRunStatus': 'Success', 'activation': 'Enabled', 'nextRun': '0001-01-01T00:00:00', 'parent': 1020, 'childScheduleItems': [], 'settings': {'schedulerSettingsId': 0, 'scheduleItemId': 1025, 'emailOnFailure': False, 'emailOnSuccess': False, 'stopOnFailure': False, 'stopToolOnTimeout': False, 'isGlobal': True, 'overrideChildSettings': False, 'successEmails': [], 'failEmails': [], 'externalToolTimeout': 0, 'enableRetries': False, 'version': {'rowVersion': 90182}}, 'version': {'rowVersion': 820550}}, {'id': 1027, 'name': 'dtTransactions data.xlsx', 'scheduleItemType': 'Import', 'order': 0, 'lastRun': '2021-07-31T17:06:22.69Z', 'lastRunStatus': 'Success', 'activation': 'Enabled', 'nextRun': '0001-01-01T00:00:00', 'parent': 1020, 'childScheduleItems': [], 'settings': {'schedulerSettingsId': 0, 'scheduleItemId': 1027, 'emailOnFailure': False, 'emailOnSuccess': False, 'stopOnFailure': False, 'stopToolOnTimeout': False, 'isGlobal': True, 'overrideChildSettings': False, 'successEmails': [], 'failEmails': [], 'externalToolTimeout': 0, 'enableRetries': False, 'version': {'rowVersion': 90182}}, 'version': {'rowVersion': 820548}}, {'id': 1028, 'name': 'Account Import Test', 'scheduleItemType': 'Import', 'order': 3, 'lastRun': '2021-07-31T17:06:24.467Z', 'lastRunStatus': 'Success', 'activation': 'Enabled', 'nextRun': '0001-01-01T00:00:00', 'parent': 1020, 'childScheduleItems': [], 'settings': {'schedulerSettingsId': 0, 'scheduleItemId': 1028, 'emailOnFailure': False, 'emailOnSuccess': False, 'stopOnFailure': False, 'stopToolOnTimeout': False, 'isGlobal': True, 'overrideChildSettings': False, 'successEmails': [], 'failEmails': [], 'externalToolTimeout': 0, 'enableRetries': False, 'version': {'rowVersion': 90182}}, 'version': {'rowVersion': 820551}}], 'settings': {'schedulerSettingsId': 0, 'scheduleItemId': 1020, 'emailOnFailure': False, 'emailOnSuccess': False, 'stopOnFailure': False, 'stopToolOnTimeout': False, 'isGlobal': True, 'overrideChildSettings': False, 'successEmails': [], 'failEmails': [], 'externalToolTimeout': 0, 'enableRetries': False, 'version': {'rowVersion': 90182}}, 'version': {'rowVersion': 820552}}

## [Part-4] Delete trigger-files

- If the process-executed with success, delete the trigger files

- trigger-files were provided in Variable3

File-deletion can be executed with a DELETE request to the following endpoint

URL = base\_URLv2 + ‘serverfiles/’+trigger-file-name /AccountTest.txt

This API call returns a status 204 when successful. And returns a 404 when it doesn’t find the file.

## Logging of activity

- Output to a log-file the activities and details of the task executions

# Setup of Logger

logging\_format = '%(asctime)s - %(message)s'

logging.basicConfig(filename=LogDoc, level=logging.DEBUG,

format=logging\_format)

logger = logging.getLogger()

logger.addHandler(logging.StreamHandler())

The following describes the log outputs we expect. Note that in green are comments.

logging.info('\*\*\*')

logging.info('\*\*\*')

logging.info('\*\*\*')

logging.info('\*\*\* Started process to Execute Scheduler Tasks’)

logging.info('\*\*\* Checking for input-variables)

logging.info('\*\*\* Process terminated: missing variable for scheduler-process-name') [if variable1 missing]

logging.info('\*\*\* Process terminated: missing variable for trigger-file-check') [if variable2 missing]

logging.info('\*\*\* Process terminated: missing variable for trigger-files') [if variable2 is True and variable 3 is missing]

logging.info('\*\*\* Process terminated: scheduler-process-name is not found in system') [if API call to obtain process-ID returns no matches]

logging.info('\*\*\* Process terminated:’+e)[if API call fails]

logging.info('\*\*\* Input-Variables were found')

logging.info('\*\*\* Checking for trigger-files’)

logging.info('\*\*\* Process terminated: trigger-files were not found') [if all trigger files aren’t found within loop window]

logging.info('\*\*\* Trigger files were found') [if all trigger files were found]

logging.info('\*\*\* Process terminated: timed-out due to another global action') [if another global action is running and loop window expires]

logging.info('\*\*\* Executed scheduler-process-name: ‘ variable1')

logging.info('\*\*\* Process completed with status of ' + < lastRunStatus>) [this is the parent’s status]

logging.info('\*\*\* Execution details: [Task1: <name>, Type: <scheduleItemType>, Execution time: <lastRun>, Execution status: <lastRunStatus>], [Task2: …], [Task3:…]') [these are the child-level details]

logging.info('\*\*\* The following files have been deleted from the sFTP: [File1name, File2name, etc]') [these are all the trigger files that have been deleted]

# Other

## Parameter File

Please ensure the below can be parameterized

### API calls

"api\_headers": {

"Content-Type": "application/json",

"Authorization": "Bearer XYZ",

"Model": "XYZ"

},

"base\_url": <https://abc.xyz.com/api/v1/>

"base\_urlv2": <https://abc.xyz.com/api/v2/>

### Duration of loops

How many minutes to perform check (X minutes)

Refresh rate of API calls in loops (may every 5 seconds)

### Output log

Enable/disable logging

Location of log\_file

Any other parameters that may be necessary.