|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DISTRIBUTION : | Firm | To | Ref | Copies | 1st page | e-mail |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

**How PTC-STARC SYNCRONIZATION API WORKS**

OBJECT

This document describes how to install and configure synchronization API between PTC and STARC

SUMMARY

/

CONCLUSION

/

|  |  |  |
| --- | --- | --- |
| STATUS: | For comments | ✓ |
|  | For application |  |

|  |  |  |
| --- | --- | --- |
| Established by | Reviewed by | Approved by |
| Name: Ardeleanu Lucian  Date: 09.02.2022  Visa: | Name:  Date:  Visa: | Name:  Date:  Visa: |

The present document contains **10 pages**, including the flyleaf and the appendices.

# EVOLUTION OF THE DOCUMENT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Issue** | **Date** | **Author** | **Rev.** | **Motive and nature of the modifications** |
| ?????? | 09.02.2022 | Ardeleanu Lucian | 1.1 | First Edition |
| ??????? | 25.02.2022 | Ardeleanu Lucian | 1.2 | Added comments and attachements description |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**CONTENTS**

EVOLUTION OF THE DOCUMENT 2

1. RELEVANT DOCUMENTS 4

2. GLOSSARY OF TERMS / EXPRESSIONS 4

3. PREREQUISITES 4

3.1. PTC Source Integrity 4

3.2. Python ( last version ) – NOT MANDATORY, OPTIONAL 4

3.2.1. USED LIBRARIES AND DEPENDENCIES: 4

3.3. Jenkins Automation Server 5

4. HOW API WORKS 5

4.1. Read Configuration File: STARC\_PTC\_Sync\_API\_Config.yaml 5

4.2. Connection to STARC and PTC databases 7

4.3. Querry PTC and STARC Databases 7

4.4. Updating the .csv file 7

4.5. Update issues according to csv file. 8

# RELEVANT DOCUMENTS

|  |  |  |
| --- | --- | --- |
| **N°** | **Title** | **Reference** |
|  | STARC\_Swagger\_API\_Use\_Cases.pdf |  |
|  | ALE ISSUE TEMPLATE V1.1\_FULL DESCRIPTION.xlsx | [I:\RBT\014\_Teams\018\_Electronics\01\_Projects\15\_SW\_Integration\01\_RBE\02\_Software\00\_Documents](file:///I:\RBT\014_Teams\018_Electronics\01_Projects\15_SW_Integration\01_RBE\02_Software\00_Documents) |
|  | SW055 A - Use of MKS for Advanced Users.doc | [I:\RBT\014\_Teams\018\_Electronics\01\_Projects\15\_SW\_Integration\01\_RBE\02\_Software\00\_Documents](file:///I:\RBT\014_Teams\018_Electronics\01_Projects\15_SW_Integration\01_RBE\02_Software\00_Documents) |
|  | SW 051C- Project and Issues management under Integrity 10 - Workflow Management.doc | [I:\RBT\014\_Teams\018\_Electronics\01\_Projects\15\_SW\_Integration\01\_RBE\02\_Software\00\_Documents](file:///I:\RBT\014_Teams\018_Electronics\01_Projects\15_SW_Integration\01_RBE\02_Software\00_Documents) |

# GLOSSARY OF TERMS / EXPRESSIONS

|  |  |
| --- | --- |
| ***Term*** | ***Definition*** |
| ***BAT*** | *Batch File ( Windows batch file extension )* |
|  |  |
|  |  |

# PREREQUISITES

## PTC Source Integrity

In order for the created API to work properly, the PTC Source Integrity software must be installed and configured correctly. Thus, this tool is used through the executable files **im.exe**, for the Integrity Manager part and **si.exe** for the Source Integrity part. In this API, the Integrity Manager part was used in particular.

## Python ( last version ) – NOT MANDATORY, OPTIONAL

**If generated .exe files from source scripts does not work** then a python envoirment is needed in order to run python scripts. It is recomented to use last version of Python in order to avoid errors.

### USED LIBRARIES AND DEPENDENCIES:

* **requests** – Python library used to send and receive requests from STARC Database.
* Json – Python library used to parse received requests from JSON in Dictionary
* Yaml – Python library used to read data from configuration file.
* Os – Python library used to connect to PTC Source Integrity with given account in yaml file
* Subprocess – Python library used to send requests to PTC through im.exe and si.exe executables
* Csv – Python library used to process csv history file.

## Jenkins Automation Server

If a continuous synchronization of the two databases is desired, it is recommended to use an automation server. This server can be, for example, a Jenkins server with a job running cyclically at a certain time interval.

# HOW API WORKS

The entire API was developed in the Python programming language. This API does not require interaction with a particular GUI, so it can be run in the background using an automation server, such as Jenkins. The configuration of the API is done ONLY through the configuration file with the extension .YAML.

**The required steps to run this API are:**

1. **Modify the STARC\_PTC\_Sync\_API\_Config.yaml configuration file** with all the data needed to run the API correctly;

2. **Launch the API: STARC\_PTC\_Sync\_API.** If you want to run it through Python, then launch the source code file with the .py extension. If you want to run the compiled API, then run the .exe file.

3. If the API worked correctly, then a .csv file should be generated, if it does not exist. If the API was previously run, then the previously generated .csv file will be updated. The history of various defects is stored in the file with the .csv extension which is automatically generated / updated.

**WARNING**!

In order to keep a history of all the defects as well as of their synchronization, data about the identified defects are stored in the csv file. Thus, if you want to resynchronize all defects from the beginning, you can delete the csv file. In order to keep the created synchronization, the csv file must also be kept.

If the .csv file is deleted and a sync has already taken place, the created API will try to remap the PTC issues with the STARC defects. Remapping may not always work properly.

The following will describe in each step how the script works.

## Read Configuration File: STARC\_PTC\_Sync\_API\_Config.yaml

The realized API will try the first step to open the configuration file named: STARC\_PTC\_Sync\_API\_Config.yaml.

**WARNING**!

The file name or extension must not be changed in any way, the file must be kept exactly as it is.In order for the realized API to be able to open the file, it must be in exactly the same path / folder as the API. ( either in .py or in .exe extension )

The configuration file structure is:

# =========================================================================

# CONFIG FILE FOR STARC PTC SYNC API

# Written by: Ardeleanu Lucian

# Date: 11 Jan. 2022

# Copyright Autoliv, RBE, 2022

# Status: Released

# ==========================================================================

# DO NOT DELETE NOWHERE ' ' CHARACHTERS!

# ---------------------------- PTC CONFIGURATION ---------------------------

# Define here USERNAME and PASSWORD for PTC Account ( Windows account )

**PTC Username**: 'firstname.lastname’

**PTC Password**: 'password'

# Write here Global path to si.exe ( if is not exist in bench )

# This paths are paths to PTC Engines

**Path to si.exe**: '"C:/Program Files (x86)/Integrity/ILMClient11/bin/si.exe" '

**Path to im.exe**: '"C:/Program Files (x86)/Integrity/ILMClient11/bin/im.exe" '

# Define here PTC Querry where to work

**PTC Querry**: 'Name of the Querry'

# Define here string that should be in issue title in order to sync it with STARC issues

**PTC Issues Title String**: '[CST]'

# PTC Project Name where issues will be created

# PLEASE MAKE SURE THAT THIS PROJECT IS SET ALSO IN YOUR PTC QUERRY

**PTC Project Name**: 'PTC\_Project\_Name\_From\_Querry'

# ----------------------------------------------------------------------------

# -------------------------- STARC CONFIGURATION -----------------------------

# Define here base URL to STARC Servers

# PRODUCTION ENVOIMENT:

**STARC Base URL**: 'https://api.starc.mercedes-benz.com'

# Define here ID and Token to connect to STARC Database

**STARC Application ID**: 'received\_id\_from\_STARC'

**STARC Application Token**: 'received\_token\_from\_STARC'

# Set here STARC Tracker Number for creating issues

**STARC Tracker Number**: "10332"

# QUERY EXAMPLES GIVEN BY STARC TEAM:

# FOR PROD ENV:

# project.id IN (18) AND tracker.id IN (10402) AND '10402.groupList[20]' IN (10459507) AND '18.10402.status' IN ('In Progress')

# project.id IN (18) AND tracker.id IN (10402)

# project.id IN (18) AND tracker.id IN (10402) AND '10402.groupList[20]' IN (10459507)

# project.name = 'body' AND tracker.name = 'Defects'

# '10332.groupList[24]' IN (10459507) or '10402.groupList[20]' IN (10459507)

# FOR INT ENV:

# project.id IN (18)

# Set here query string in cbQL Language

**STARC cbQL Querry**: "querry\_name"

# Set this status to True if you want to sync only items with Sync Allowed set to Yes in STARC Items

**Sync Allowed Status**: False

# -------------------------- GENERAL CONFIGURATION ---------------------------

# Define here path to csv file that contains hisory of issues

# Path must contain also filename and extension

**Path to csv history file**: 'STARC\_PTC\_Sync\_History.csv'

# ----------------------------------------------------------------------------

# Other useful information:

# ISSUE FIELDS FROM CSV ARE:

# issue[0] -> STARC Issue Number,

# issue[1] -> PTC Issue Number,

# issue[2] -> Last STARC Issue State,

# issue[3] -> Last PTC Issue State,

# issue[4] -> Current STARC Issue State,

# issue[5] -> Current PTC Issue State,

# issue[6] -> STARC New Issue number assigned to PTC issue,

# issue[7] -> STARC\_Issue\_Title,

# issue[8] -> PTC\_Issue\_Title

## Connection to STARC and PTC databases

The connection to both databases is made using the data received from the configuration file. Thus, in the PTC database, it is necessary to connect using a username and a password.

Within the STARC database, it is necessary to connect to the STARC API link using an ID and a token. If the connection was successful, then a unique ID corresponding to the current session will be generated. Through that ID, it will still be possible to send various requests to STARC.

If the connection did not take place in one of the two databases, the user will be notified via a console error.

## Querry PTC and STARC Databases

If the connection is successful, a query is made on each database.

Thus, in the case of the PTC database, the query is made based on the Query specified in the configurations, and from the respective query are taken ONLY the issues that contain a certain string in the title, a string that was also specified within the config file. (eg [CST] -> Customer) The software function that will perform the respective query will take the name of the issues, their ID and their State and will return a list of lists, each element of the list containing a issue identified.

Within the STARC database, this software function takes over the session ID generated at login, the link where requests are sent specified in the config file and the cbQL query also specified in the config file. This function will return a dictionary with all defects that meet the requirements of the query.

## Updating the .csv file

After performing the queries on both databases, you can start updating or generating the historical file with the .csv extension as needed.

It was decided that this file should have a tabular structure, with the following columns:

column 1 -> STARC Issue Number, is the id of the issue identified in STARC

column 2 -> PTC Issue Number, is the issue id identified in PTC

column 3 -> Last STARC Issue State, is the last STARC Issue Status (different from the new one)

column 4 -> Last PTC Issue State, is the last PTC Issue State (it is different from the new one)

column 5 -> Current STARC Issue State, is the current status of the STARC issue

column 6 -> Current PTC Issue State, is the current status of the PTC issue

column 7 -> STARC New Issue number assigned to PTC issue, this column is currently unused.

column 8 -> STARC\_Issue\_Title, is the title of the STARC issue

column 9 -> PTC\_Issue\_Title, is the title of the PTC issue.

The csv is updated through two software functions, each function will update the data from a certain database.

Both software functions will read the csv file, check if the issue found in the previous query is already in the database or not. If it is, then it is checked if it has a different State compared to the State found in the query.

If the State differs, then the State from csv in column 5 or 6 will move to the positions in column 2 or 3, depending on the software function of the database. Columns 5 or 6 will pass the new identified State from the database querry.

If the State remains the same, then no change is made.

Also, in the software function that updates the STARC issues in the .csv file, it is checked if a certain string has been identified in the issue description, a string that could contain a PTC issue ID. If it has been identified, then the STARC issue ID from csv is remapped with the newly identified query ID. This update was decided because at the time of developing this API, the STARC database differentiates the issues received from the customer and creates another issue based on them. In fact, it is necessary that the new issues created by STARC keep in their description the issue ID of the PTC on the basis on which it was created.

If the issue found in the query was not identified in the csv, or, in other words, a new issue appeared and is not in the csv, then the csv is updated with the new issue.

## Update issues according to csv file.

After the csv file has been updated, the issues can be updated according to the csv.

Thus, 3 cases are possible:

**CASE 1 - A NEW ISSUE ARRIVED FROM STARC AND HAS NOT A PTC ISSUE ASSIGNED:**

In this case, a new issue must be created in the PTC. By iterating in the issue dictionary and their property received from the query made above, the following fields are taken into account:

1. **Item name** - is taken from STARC and a certain string is added to it according to the requirements of the PTC.

2. **Item description** - is taken from STARC and put in the description of the new item from PTC

3. **Item ID from STARC** - is taken from STARC and placed in the Customer Isssue ID section of PTC

4. **The priority of the STARC issue** - it is taken over and mapped according to the STARC id in PTC

5. **The severity of the issue in STARC** - is taken and mapped as follows: High and Ultra High will have the severity mapped Major in PTC, Medium and Low will have the severity mapped Minor in PTC, and Not Applicable and Unset will have the severity mapped Change in PTC

6. **The STARC Release field** will be taken over, more precisely the **Detected in EE Release field** and will be filled in the Keywords section of the PTC.

Also, comments and attachements will be synchronized for each STARC Item in PTC Item. Backward synchronization between comments and attachements from PTC to STARC has not yet implemented due to security access on STARC account.

After creating the item, its ID is taken over and filled in the csv file, according to the issue from which it was created.

**CASE 2 - A NEW ISSUE ARRIVED FROM PTC AND HAS NOT A STARC ISSUE ASSIGNED:**

In this case it is necessary to create a new item in STARC. The following fields are taken from the PTC item:

1. **PTC item ID** - is taken over and added to the new STARC item description along with the string: 'AUTOLIV ISSUE ID:'

2. **PTC item title** - is taken and passed as the item title sent to STARC

3. **PTC item description** - is taken over and added to the new STARC item description

4. **Severity of the PTC item** - is taken over and mapped in STARC as follows: if the severity is Major, then a High severity is specified in STARC, if the severity is Minor, a Low severity is specified in STARC, and in any other case it is specified Not Applicable STARC severity

5. The assigned person is specified in the **Assigned To field**.

6. The **Occurrence field** specifies the occurrence Sometimes.

7. The value No is specified in the **Veirification By Test Group field**

8. The priority of the PTC item is taken over in the **Priority field**.

After creating the item, its ID is taken over and filled in the csv file, according to the issue from which it was created.

**CASE 3 - IF ISSUES ARE ALREADY CREATED, THEN CHECK STATES**

In this case, only the mapping of the States fields is performed, and the updates of the title, description or other fields are ignored.

3 cases are possible:

In the first case it is checked if **STARC CURRENT STATE IS IN PROGRESS, LAST STARC STATE IS FIXED AND CURRENT PTC STATE IS CLOSED** then the item from STARC is put in the Fixed Status.

In the second case it is checked if **STARC CURRENT STATE IS IN PROGRESS AND PTC CURRENT STATE IS REJECTED** then the STARC item is put in the Open Status.

In the third case it is checked if **STARC CURRENT STATE IS IN PROGRESS AND STARC LAST STATE IS FIXED** then the item from PTC is put in the State (To Be Analyzed).

After updating the issues, the script will notify the user in the console of each step it has taken.