MCA Configuration Management Automation

Implementation Document

V1.0

## Purpose

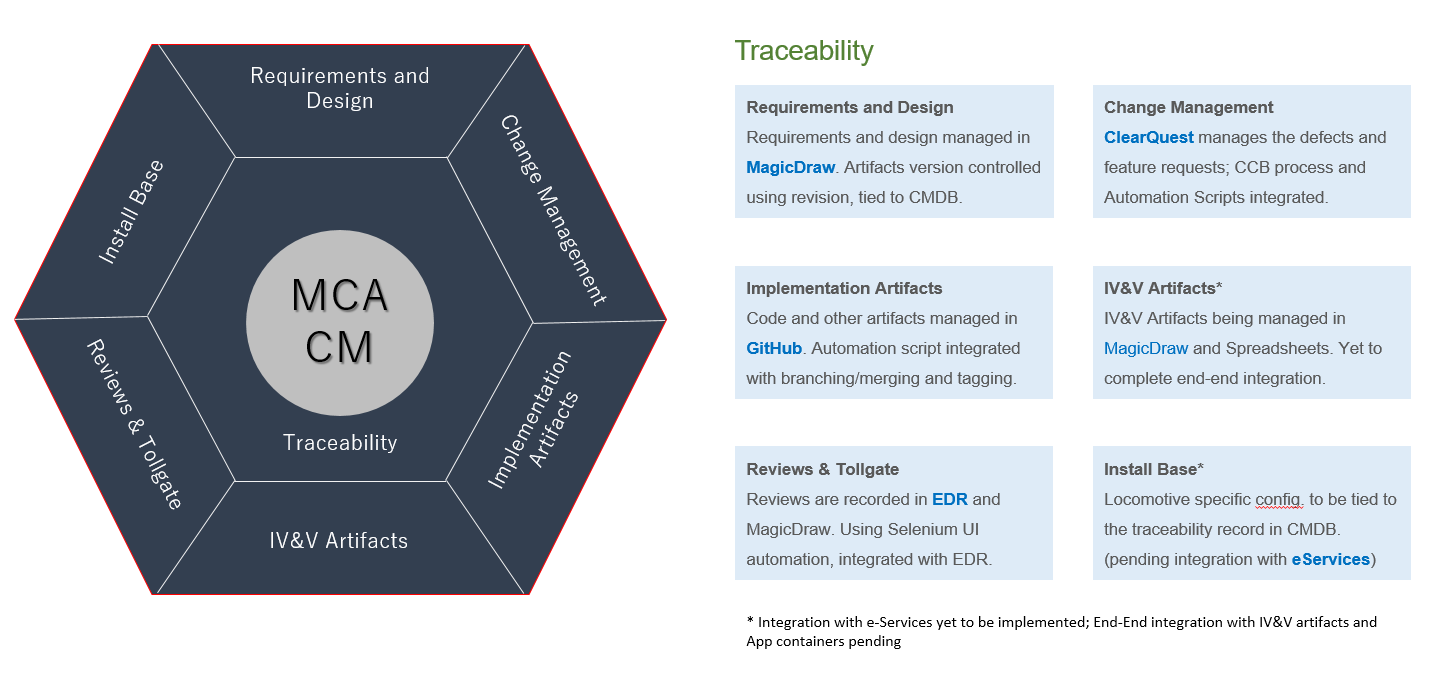
The purpose of this document is to detail the implementation details and process flow of the automation scripts that are used in the MCA Configuration Management.

## Scope and Goals

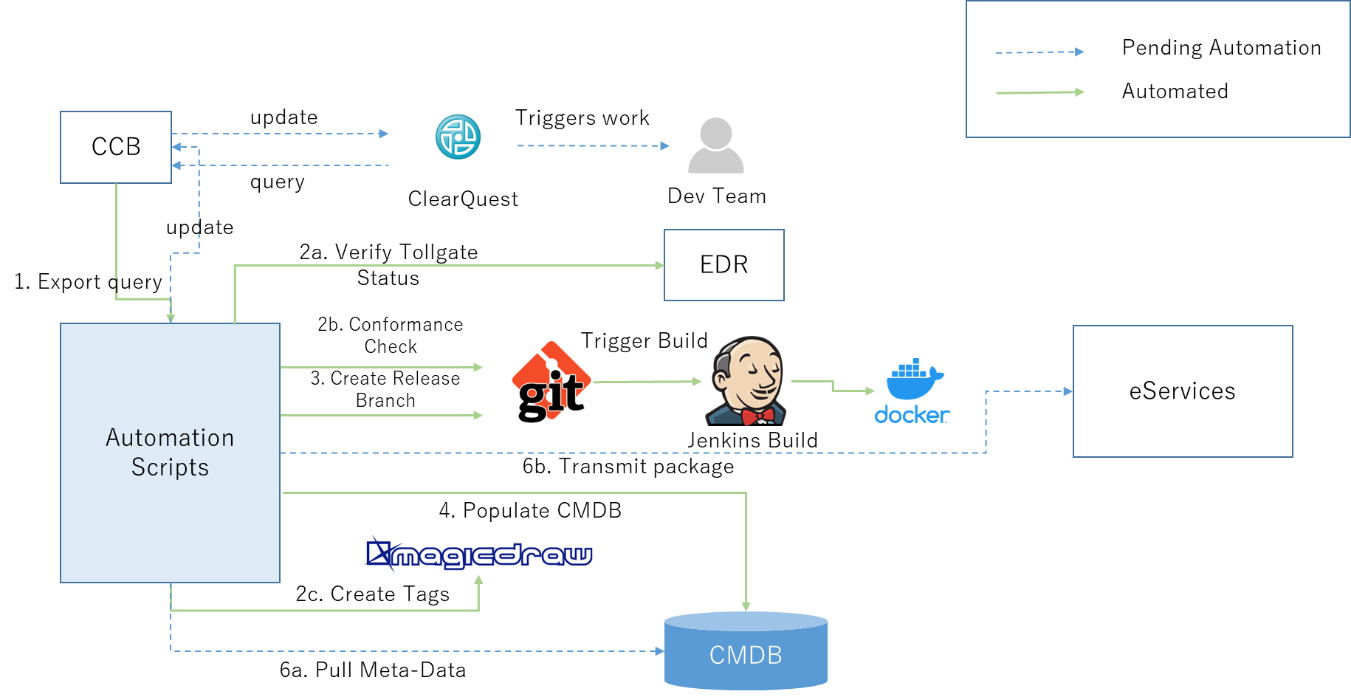
The MCA CM automation is aimed at bringing traceability across the lifecycle to cover the following areas:

* Requirements and Design
* Change Management
* Implementation Artifacts
* IV&V Artifacts
* Review and Tollgate
* Install Base

The overall scope is depicted below:



## High Level Solution View



## Implementation

The automation is implemented with modular components and scripts. The following are the scripts used in the automation:

1. cmdbdesign.sql
2. insertscript.sql
3. masterscript.py
4. cmdbinsert.py
5. Edrstatus.py
6. release-insert.py
7. Jenkinsupdate.py

CMDB connection details,

Connect to 3.125.182.23 server, there you can use the root user and root password to connect,

mysql -u root -proot;

show databases;

use cmdb\_design;

show tables;

select \* from cmdb\_design.repo;

select \* from cmdb\_design.repocommits;

1. ***cmdbdesign.sql:***

This script will create and setup the CMDB database. The CMDB tables along with the purpose are listed below:

1. RELEASE
   * This table will have Release information including release branch name, date and time and some static information like URLs for the Jenkins, GITHUB, EDR, Confluence and etc. which will pull this information from the insertscript.sql.
   * Release ID is the Primary Key in this table.
   * This table is the key parent table that ties all the artifacts together.
2. REPO
   * This table will contain the information about the repo and the commit number that are used to branch for that release.
   * The information is got by pulling from the ClearQuest export Excel sheet that will have the information of repo names, latest commit number, Release Type and Jenkins status.
   * For example, if there were 3 commits in a single repo for a release, this table will only contain the latest commit number that is used to branch. Rest of the Commit and repo details can be found in REPOCOMMITS table.
3. SCO
   * This table will have the information of SCO #. (Future use once ClearQuest API is available)
4. DEFECT
   * This table will have information of the SCO and SRP. (Future use once ClearQuest API is available)
5. REPOCOMMITS
   * This table will have the information of the all the Repos and the Commits
   * This will include the repos with the same name but multiple commits present in the CQ Export Excel sheet, EDR number and its status.
6. AUDITLOGS
   * This table will have the information of Log level info, log message and log time stamp.
   * This is used to troubleshoot various scripts’ functionality.
7. PROPERTIES
   * This table will have the static information like property name and property value (e.g., Build Machine URL, Toll Gate URL, updated by and update time) pull from the *insertscript.sql* file.
   * This table will avoid hard-coding of various parameter values in the script.

The current CMDB ER Model is shown below:

A screenshot of a cell phone

Description automatically generated

1. ***insertscript.sql:***

This DML SQL script will have the information of static information of Build Machine URL, Toll Gate URL. This script can be enhanced to maintain all master data and Properties table insert scripts.

1. ***masterscript.py***

Master script is the main script and the only script you will run for each release. The script will first look for the QueryResult.xls file in the folder. Then it will convert the XLS file extension into CSV to read the information of the defects. It will look for the DEF\_Changes list field in the ClearQuest Export Excel sheet to confirm whether the defect belongs to Git or MagicDraw.

If it is Git, it will check the Product\_Descriptors field in the ClearQuest Export Excel sheet to verify the defect belongs to MCA Platform or MCA Applications. Once the project is verified then it will look for the repo name from Product\_Descriptor\_Level3 field. Then, after it will check for the repo type. We have four kind of repos under GitHub platform organization. They are Service based repo, COTS repos, Docker build image repos and library repos. We only need release branch for service based repo. For the remaining repo types, we need to merge develop with the master branch. So the script will identify the type of repo and it will create the release branch for Service based repos. For non-Service Based Repos, it will merge the develop branch with the master. The script will take the repo name from Product\_Descriptors\_Level3 field and based on the version number in the Def\_ChangesList field, it will create the release branch also pre-release tag using GitHub APIs programmatically.

If it is MagicDraw item (defect/feature), the script will tag the MagicDraw project. The script takes the version number from the Def\_Changeslist field in the ClearQuest Export Excel file and looks for the Project name from Product\_Descriptors. There are four different projects in MagicDraw, namely MCA Platform SW, MCA Systems, MCA IV&V and MCA Applications. These four project have the corresponding project ID. So based on the project mentioned in the Product\_Descriptors, the script will take the ID from the Projected.py python script and will tag the given version by using MagicDraw APIs with the same name that is used to create the release branch.

**Note**: In order for the script to function properly, the work instructions given by the CM team must be followed strictly. Check the work instructions document [mca-cfg-cqwi\_defects.docx](https://devcloud.swcoe.ge.com/devspace/download/attachments/1532307676/mca-cfg-cqwi_defects.docx?api=v2) here <https://devcloud.swcoe.ge.com/devspace/pages/viewpage.action?pageId=1532307676>.

python masterscript.py

The script will request the user to enter the release branch name: IVV-PLT-2020.0317.1200 (For the consistency, please follow this format IVV-PLT-YEAR.DATE.TIME).

1. ***cmdbinsert.py:***

When you run the masterscript.py, it will also run two scripts one of them is cmdbinsert.py. The cmdbinsert.py script reads the initial values (like Repo name, Latest commit, Release type and Jenkins status, Jenkins status for Git items from the QueryResult.xls file and inserts into the REPO table and REPOCOMMITS table. *Cmdbinsert.py* scripts triggers the next python script called edrstatus.py.

1. ***edrstatus.py:***

This script will run with the cmdbinsert.py. The EDR status will be obtained using Chrome Headless automation based on the EDR number. In future, this script shall be modified to use EDR APIs or Database based integration to get the status. Once the status is obtained, it will be inserted into the REPOCOMMITS table for every applicable REPO Commit. The EDR number input will be taken from ClearQuest Export Excel file’s DSRVW # field.

1. ***release-insert.py:***

This script will be triggered by the master script. This script will insert the values into the RELEASE table.

1. ***jenkinsupdate.py:***

Once the creation of release branch from Git is done, it will trigger the Jenkins build. After the successful build, the build script will push the docker image to the DTR. The Jenkins status and DTR information is passed back to this script from the Jenkins Build file. This script will take those arguments from the Jenkins build script and it will UPDATE those values into the REPO table. This script is not integrated yet with the Jenkins file. This this script can also be run manually. But, when run manually, the following parameters must be passed: Branch Name, Repo Name, DTR and Jenkins status. The script can be run by using the following command:

Python jenkinsupdate.py BRANCH\_NAME REPO DTR JENKINS\_STATUS

E.g., jenkinsupdate.py IVV-PLT-2020.0317.1200 gets-filetransferservice IVV-PLT-2020.0317.1200.1 Success

The following flowchart explains the interactions between the various scripts.

A close up of a map

Description automatically generated