XL-Migration-Tool

April 8, 2019

# Purpose and Description of the Tool

The XL-Migration-Tool is designed to update XL Release database and by doing so efficiently migrate an XL installation from the use of an outdated version of a plugin to the use of a newer version of that plugin.

The tool is a stand-alone java program that performs actions directly upon the database and is driven by an external JSON mapping file. The SQL statements used to perform the actions are defined within the application and are database type specific.

The tool is used outside of XL Release. The XL Release application cannot be running at the time of migration.

Dependent upon information in the external JSON mapping file, migration will accomplish ONE of the two following goals when run against an installation that has been previously configured with an outdated plugin:

1. Goal – After migration, the installation will run error free with only the new plugin installed. The tool updates the database, migrating all old configuration information to work with the new plugin.

# What the XL-Migration-Tool Does

The current implementation of the tool has been designed to perform the following functions, according to the actions configured in the mapping file:

* XL Release
  1. Update Task - In the XLR database - change the type and properties of a TASK by updating the content (blob) of the XLR\_RELEASES\_DATA table, update task properties in the content (blob) of the XLR\_TASK\_BACKUPS table, update the task\_type in the XLR\_TASKS table. In the XLR Reporting database - update the task properties releasejson (blob) in the RELEASES table, update the tasktype column in the TASKS table.

All operations can also be run in preview mode. Preview mode queries the XLR\_CONFIGURATIONS and XLR\_TASKS tables to report the count of items that fit the criteria for each action in the mapping file. The database is not changed.

# How the XL-Migration-Tool works

When run, the tool first parses the XL installation XL Config file (xl-release.conf) to discover if the XL installation is using an embedded or external database. For XL Release installations, the tool also parses information about the reporting database. If no external database information is discovered, the default embedded database information is used. The tool dynamically loads necessary database driver jars from the XL installation lib directory.

NOTE: If databases have been configured with passwords, the user of the XL-Migration-Tool can be prompted to supply database passwords. The command line entry of database passwords is masked. The tool does not use password information stored in the xl-release.conf files.

The XL-Migration-Tool then parses the external JSON mapping file to configure the tool actions.

Configured actions are run against the database or databases. Each action may require multiple SQL statements, all run in a single transaction per action. If a specific action fails, the action transaction is rolled back and the tool exits. After the cause of the error is fixed, the entire mapping file can be run again. Successfully completed actions, committed before the tool exited, will be ignored and the problem action will be tried again. The tool will update data stored in directly in table columns and will update configuration data stored in json format within blobs.

# How to use the XL-Migration-Tool

The XL-Migration-Tool is a stand-alone java application that includes most dependencies within the jar itself. The only other jars needed are database drivers which the tool will automatically discover and dynamically load. The tool is run at the command line. Arguments, both required and optional, are listed below:

1. -f Required - Supply the full path and file name for the mapping file.
2. -i Required - Supply the full path to the XL Release installation directory. Please note: as mentioned above, the XL-Migration-Tool will load database driver jars dynamically. The tool uses the installation directory to find the lib directory within the XL installation and loads all jars from there. The tool also uses the installation directory to find embedded databases in the case where external database have not been configured.
3. -pw Optional - If this flag is set, user will be prompted for the database password.
4. -reportpw Optional - If this flag is set, user will be prompted for the report/archive database password.
5. -preview Optional - If this flag is set, the application will only preview the mapping actions. The database will not be changed.

Example:

# java -jar xl-migration-tool-1.0.jar -f "xlr-servicenow-pluginMigration.json" -i "./xl-release-8.6.1-server"

# XL-Migration-Tool Source Code

A gradle build file is included in the source code. To build the jar, run gradle fatJar. A mapping file – xlr-servicenow-pluginMigration.json - is also included and can be found in servicenow directory. This mapping file has been configured to perform the following actions:

**For all Update TASK actions**, it will move entries from ‘content’ to new ‘Additional fields’, and the ‘short\_description’ within the ‘content’ field to the new ‘Short Description’ field.

1. Update TASK – Change CreateRecord to the new CreateChangeRequest
2. Update TASK – Change CreateNewRecord to the new CreateChangeRequest
3. Update TASK – Change UpdateRecord to the new UpdateChangeRequest
4. Update TASK – Change CreateNewIncident to the new CreateIncident
5. Update TASK – Change CreateNewServiceRequest to the new CreateServiceRequest
6. Update TASK – Change CreateNewRequestItem to the new CreateRequestItem
7. Update TASK – Change CreateNewChangeRequest to the new CreateChangeRequest
8. Update TASK – Change CreateTask to the new CreateChangeTask
9. Update TASK – Change UpdateTask to the new UpdateChangeTask
10. Update TASK – Change CreateRequest to the new CreateChangeRequest
11. Update TASK – Change UpdateIncident to the new UpdateIncident
12. Update TASK – Change UpdateServiceRequest to the new UpdateServiceRequest
13. Update TASK – Change UpdateRequestItem to the new UpdateRequestItem
14. Update TASK – Change UpdateChangeRequest to the new UpdateChangeRequest

# The XL-Migration-Tool Testing Scenario

1. Back up of the current repository and archive database was done.
2. XL Release installation with the new version of the plugin installed, using the embedded, default databases.
3. A template, containing all old version tasks was created.
4. A release, using the template, was started and upon failure, was aborted. This was done to ensure that the release would be placed in the archive/reporting database.
5. After the aborted release was archived, XL Release was shut down.
6. The XL-Migration-Tool was run, using the xlr-servicenow-pluginMigration.json mapping file.
7. XL Release was restarted.
8. The Template, Template Tasks and archived Release were all displayed within the UI to ensure XL Release threw no exceptions.
9. Template Tasks were examined to ensure properties had been properly migrated.
10. The migration has not been tested against other supported databases but the necessary configuration for MySQL, Postgresql, Oracle, Db2, and SQLServer are in place, ready for testing.