XL-Migration-Tool

September 19, 2018

# Purpose and Description of the Tool

The XL-Migration-Tool is designed to update either an XL Deploy database or XL Release databases and by doing so efficiently migrate an XL installation from the use of an outdated version of a plugin to the use of a newer version of that plugin.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 Deploy or XL Release. The XL Deploy or 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accomplish ONE of the two following goals when run against an installation that has been previously configured with an outdated plugin: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.

- Or -

2. Goal – After migration, the installation will run error free, without namespace collision, with both a variant of the old plugin and the new plugin installed. The variant of the old plugin is created by altering the synthetic.xml (and some object names within code) with new namespace information. The tool is used to update the namespace of previously configured items within the database.

# 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 Deploy
  1. Update CI - Change the type of a CI by updating the ci\_type column in the XLD\_CIS table
* XL Release
  1. Update CI - In the XLR database - The tool will change the type and properties of a CI by updating the ci\_type column and the content (blob) column in the XLR\_CONFIGURATIONS table.
  2. 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 XLD-CIS table (in the case of XL Deploy) or the XLR\_CONFIGURATIONS and XLR\_TASKS tables (in the case of XL Release) 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-deploy.conf or 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-deploy.conf or 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.

# Current xlr-xld-plugin Migration Strategy Compared to Using the XL-Migration-Tool

According to <https://docs.xebialabs.com/xl-release/how-to/xld-plugin-community-to-official.html>, the current procedure is as follows: User must define new Servers and CLI Configurations and then must manually migrate all effected tasks using the "Change Task Type" feature. Many task properties must then be manually reconfigured. Both the community and the official plugin must always be installed, even after migration.

Alternatively, the XL-Migration-Tool will remove most of the manual migration steps. Shared Servers and CLI Configurations are programmatically migrated, task types are migrated, and most task properties are migrated. After migration, the old community plugin should be removed. This eliminates the confusing display of similarly named tasks in the UI and the possibility that users will mistakenly choose deprecated tasks when configuring new templates.

In the case where a specific task property cannot be migrated, the tool user is notified that a manual configuration step must be performed after migration.

For this project there is only one task property that the tool cannot migrate:

1. For the task (XL Deploy - Deploy Task) - the Deployed Properties property cannot be migrated and must be reconfigured after the migration tool runs. Reason: The old plugin supported an XPath value for this property and the new plugin does not.

There are a few tasks that we believe can be migrated but have not been added to the mapping file because they have not yet been implemented in the official version of the plugin. Once that work has been completed, we can add the mappings, test and make changes to the tool, if necessary. Those tasks are listed below:

1. OLD TYPE - xldeploy.CreateEnvironment
2. OLD TYPE - xldeploy.CreateDictionary
3. OLD TYPE - xldeploy.CreateSimpleCI
4. OLD TYPE - xldeploy.XLDVersionsTile

# 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 Deploy or 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 "/Users/tester/XLR-ALL.json" -i "/xl-release-8.1.0-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-xld-pluginMigration.json - is also included and can be found in src/resource directory. This mapping file has been configured to perform the following actions:

1. Update CI - migrate existing type server to new type server
2. Update CI - migrate existing type CliConfig to new type CliConfig
3. Update TASK - migrate existing task type DeployTask to new task type
4. Update TASK - migrate existing task type UndeployTask to new task type
5. Update TASK - migrate existing task type ControlTask to new task type
6. Update TASK - migrate existing task type Migrate Package Task to new task type
7. Update TASK - migrate existing task type Import Package Task to new task type
8. Update TASK - migrate existing task type Get Latest Version Task to new task type
9. Update TASK - migrate existing task type Get All Versions Task to new task type
10. Update TASK - migrate existing task type Get CI Task to new task type
11. Update TASK - migrate existing task type Get Latest Version Deployed Task to new task type
12. Update TASK - migrate existing task type Does CI Exist to new task type
13. Update TASK - migrate existing task type Create Folder Tree to new task type
14. Update TASK - migrate existing task type Create CI to new task type
15. Update TASK - migrate existing task type Delete CI to new task type
16. Update TASK - migrate existing task type Delete Infrastructure to new task type
17. Update TASK - migrate existing task type UpdateCIProperty to new task type
18. Update TASK - migrate existing Cli Run Script task type to new task type
19. Update TASK - migrate existing Cli Run Script File task type to new task type
20. Update TASK - migrate existing Cli Run Script URL task type to new task type
21. Update TASK - migrate existing task type Get CI Tags to new task type
22. Update TASK - migrate existing task type Add CI Tag to new task type
23. Update TASK - migrate existing task type Set CI Tag to new task type

# How the XL-Migration-Tool Testing Scenario

1. XL Release installation, with both the old version and the new version of the plugin installed, using the embedded, default databases.
2. An old version of both the XL Deploy Server and the XL Deploy CLI Configuration was created.
3. A template, containing all 21 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-xld-pluginMigration.json mapping file.
7. Within the XL Release installation, the old xlr-xldeploy-plugin-3.7.0.jar was deleted.
8. XL Release was restarted.
9. The XL Deployed Server, XL Deploy CLI Configuration, Template, Template Tasks and archived Release were all displayed within the UI to ensure XL Release threw no exceptions.
10. Template Tasks were examined to ensure properties had been properly migrated.
11. 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.