**ExxonMobil Chemical Company**

**Application Migration**

**User Manual**

**Version 0.5**

**Revision Date: October 22, 2015**

**Prepared by:**

**ILS Automation Inc.**

REVISION HISTORY

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description |
| 0.1 | January 15, 2015 | CC | Initial draft |
| 0.2 | February 12, 2015 | PH | Reformatted, Added: Export from G2 details; Sequential Control Callback details |
| 0.3 | June 17, 2015 | PH  CC | Added section about tag export / import files.  Added commentary re: auto-generated python |
| 0.4 | September 8, 2015 | PH | Added Lab Data discussion. |
| 0.5 | October 22, 2015 | PH | Revised the I/O tag creation section to reflect a streamlined process |

REFERENCES

|  |  |  |
| --- | --- | --- |
| Title | Author | Version |
| ExxonMobil – BRCP – Toolkit Migration Project | ILS Automation | 2.0r1 |
| Application Test Framework | ILS Automation | 0.2 |

TABLE OF CONTENTS

[1 Introduction 5](#_Toc433321927)

[2 I/O 5](#_Toc433321928)

[2.1 OPC Server Configuration and Translation 5](#_Toc433321929)

[2.2 Recipe Data I/O 5](#_Toc433321930)

[2.3 Lab Data I/O 6](#_Toc433321931)

[2.4 Diagnostic Toolkit I/O 7](#_Toc433321932)

[2.5 Sequential Control I/O 8](#_Toc433321933)

[2.6 Diagnostic Toolkit Connections 8](#_Toc433321934)

[2.7 Manually Created I/O 8](#_Toc433321935)

[2.8 Delivery of Tag I/O Configuration 9](#_Toc433321936)

[2.9 Tag Structure in Ignition 10](#_Toc433321937)

[3 Recipe Toolkit 10](#_Toc433321938)

[3.1 Create Trigger for Automatic Download 10](#_Toc433321939)

[4 Diagnostic Toolkit 11](#_Toc433321940)

[4.1 Export from G2 12](#_Toc433321941)

[4.2 Tag Mapping 12](#_Toc433321942)

[4.3 Convert 13](#_Toc433321943)

[4.4 Calculation Procedure Translation 13](#_Toc433321944)

[4.5 Import into Ignition 14](#_Toc433321945)

[5 Sequential Control 14](#_Toc433321946)

[5.1 Export from G2 14](#_Toc433321947)

[5.1.1 Callback Export 14](#_Toc433321948)

[5.2 Convert 17](#_Toc433321949)

[5.2.1 Procedure Conversions 17](#_Toc433321950)

[5.2.2 Block Conversions 17](#_Toc433321951)

[5.2.2.1 Parallel Transitions 17](#_Toc433321952)

[5.2.3 Recipe Data Conversions 18](#_Toc433321953)

[5.2.3.1 Simple Data Sequence Type 18](#_Toc433321954)

[5.3 Import into Ignition 18](#_Toc433321955)

[5.3.1 Callback Procedures 18](#_Toc433321956)

[5.3.1.1 Python Location 18](#_Toc433321957)

[5.3.1.2 Long Running Callbacks 18](#_Toc433321958)

[6 Lab Data Toolkit 19](#_Toc433321959)

[6.1 Lab Data 19](#_Toc433321960)

[6.2 Lab Feedback Control 19](#_Toc433321961)

# Introduction

A major component of the ExxonMobil Chemicals Ignition Applications project is the translation of advanced control applications from their current G2 platform onto Ignition™, a product of Inductive Automation. Ignition is an extensible platform that provides OPC connectivity, database integration and a graphics library for interface development. It is designed to be extensible via Python scripting and custom Java modules. These features are used extensively during the migration.

Due to the large number of applications involved, it was essential that automated facilities be developed to handle the conversions. To the extent possible, the migration tools run in a “hands-off” mode, that is, scripted.

This document is a “how to” guide for the operation of those tools.

# I/O

One of the major efforts during a site migration is configuring the I/O. Each of the toolkits uses a slightly different technique to migrate the necessary I/O. Regardless of which toolkits are used at a site, the first step is to configure the OPC interfaces and scan classes manually and to configure the database table that translates from the old G2 GSI interface names to the new names.

The process of creating all of the tags necessary for a site will generally take a number of weeks and several migrators may be involved. Ultimately, a single xml export sqlTags.xml will be delivered to the site that contains all of the tags, but during the process it is important to keep the tags in several “import” files so that the complete set can be built from its constituent parts.

## OPC Server Configuration and Translation

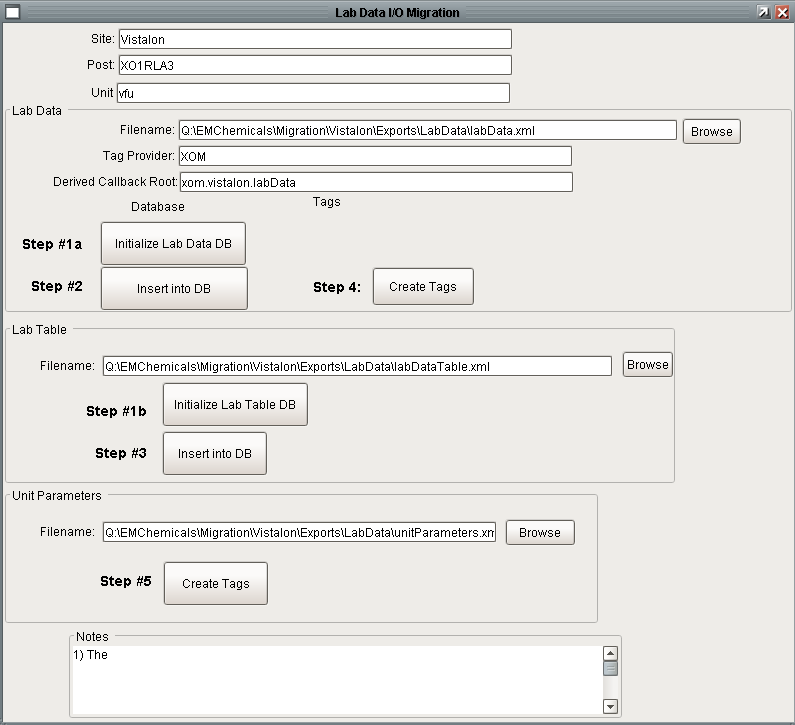
To be determined.

## Recipe Data I/O

There are two types of I/O used for Recipe Data. The first is named memory tags that are internal to Ignition. These are created in a hand crafted script that runs on startup. The necessary tags are determined by inspecting the recipe database for tags whose location is g2 or local. The second type of I/O is actual OPC outputs implemented via UDTs. These are created programmatically on the fly when the recipe download screen is loaded. As part of migration, a recipe download can be simulated on the development server which will create the tags and place them in the Recipe folder.

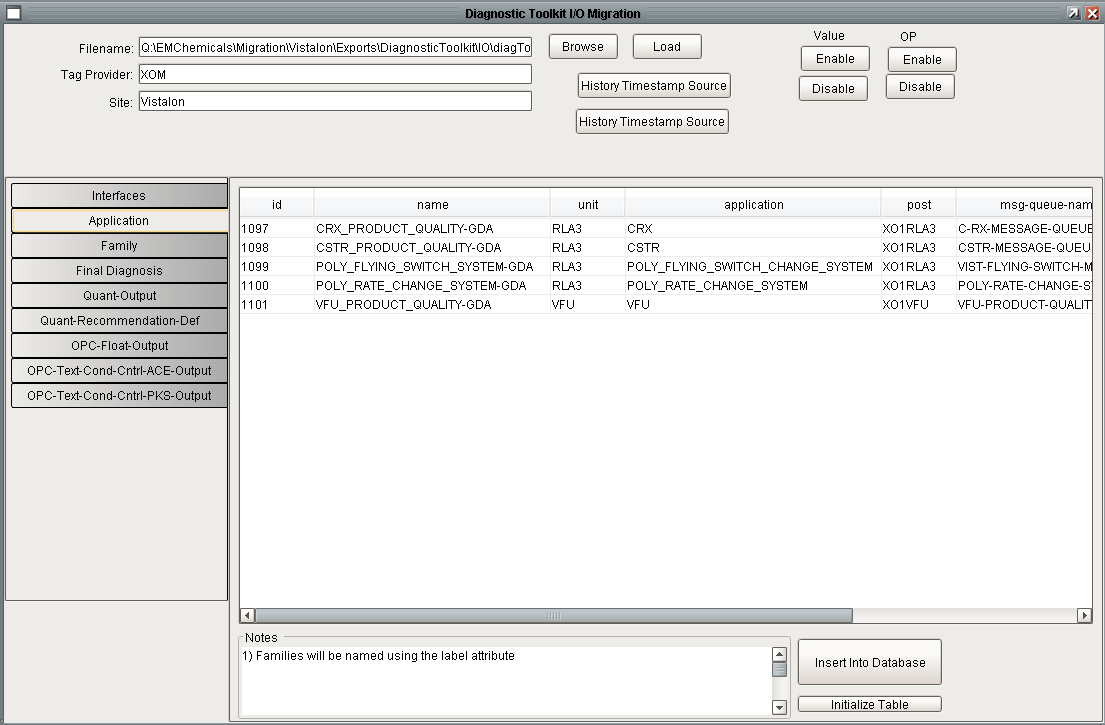
## Lab Data I/O

Lab Data I/O are created by running the lab data export utilities in G2 and then using the migration window in Ignition shown below:



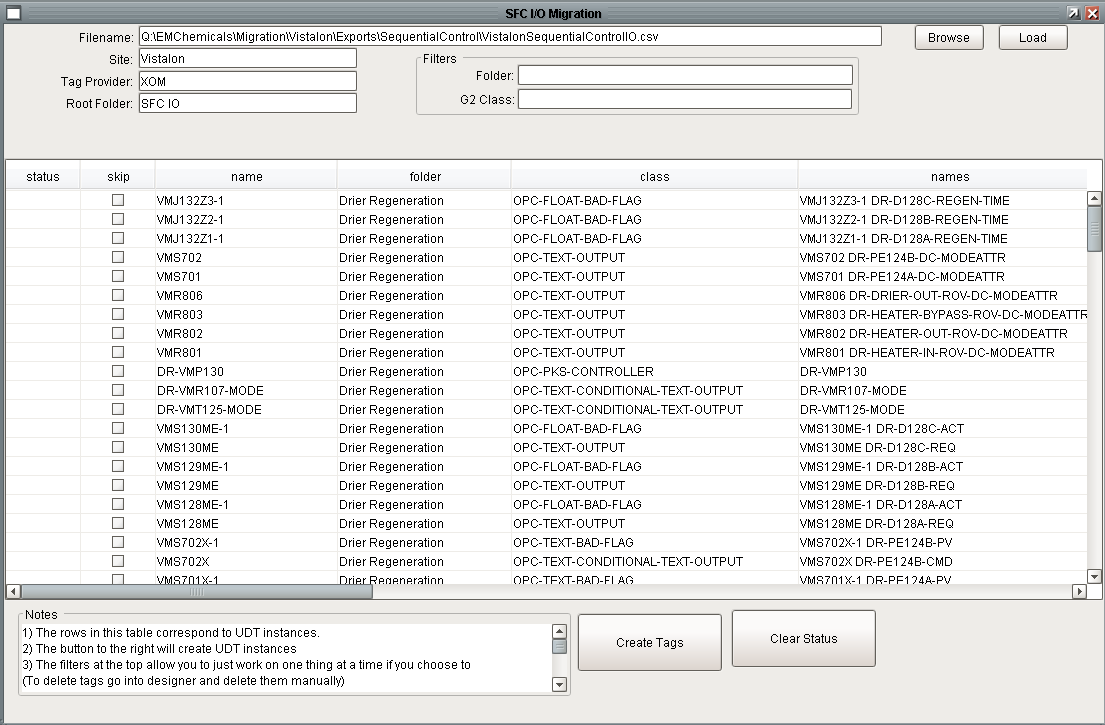
## Diagnostic Toolkit I/O

Generally, the majority of the inputs for the Diagnostic applications come from Lab data and will have been created in the previous step. The outputs are created from a custom migration screen available in the online application, shown below:



## Sequential Control I/O

Like Lab Data and Diagnostic Toolkit I/O, the process of creating I/O for the Sequential Control toolkit begins with exporting the configuration from G2. The export file should be checked into SVN in the site specific migration export folder. The export file is loaded into Ignition and tags created using the screen shown below.



## Diagnostic Toolkit Connections

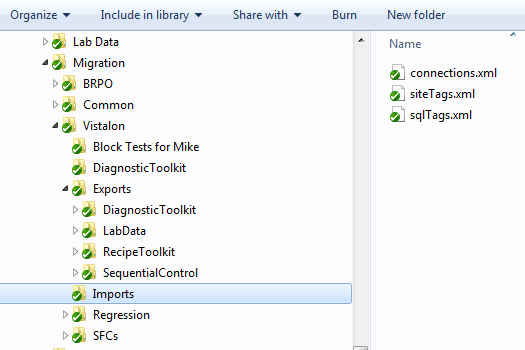
Tags are used to implement connection posts and facilitate communication between diagnostic diagrams. These tags are not exported directly from G2, rather they are created by the BLT G2 converter utility which reads the diagram export files. The file is imported by the migrator into the development system to facilitate off-line testing.

## Manually Created I/O

In addition to all of the I/O created by the above methods, there will undoubtedly be additional I/O that must be created. In order to provide a rebuild from a clean sheet, the tags should be defined in an XML file. This will allow the complete set of tags to be created on a newly initialized system from scratch following the steps described above. The I/O in this file will generally be hand created by the migrator as unresolved G2 tag references are discovered after all of the automated utilities are run. This file may contain tags that touch all of the specific toolkit folders, therefore it should be imported after all of the automated tools have been run.

## Delivery of Tag I/O Configuration

Creation of the complete set of I/O for a site is the responsibility of the site migrator. The majority of the migration effort is performed off-site on ILS computers. The utilities described above are executed on the migrators computer. As such, the migrators computer becomes the master repository of a site’s I/O. In order to deliver the I/O configuration to the site, the tag definitions are exported in an XML file titled sqlTags.xml. This file contains all of tags used by all of the toolkits at the site and is the only file that needs to be imported at the site. The folder structure that contains the files described above is shown below.



A summary of the files is:

| File | Description | Auto or Manual |
| --- | --- | --- |
| connections.xml | Created by the BLT G2 converter export/import utility. | Auto |
| siteTags.xml | Hand created by the migrator as unresolved G2 tag references are discovered after all of the automated utilities are run. This file may contain tags that touch all of the specific toolkit folders, therefore it should be imported after all of the automated tools have been run. | Manual |
| sqlTags.xml | The complete set of tags required by the site. This is created by the migrator by manually exporting the tags from Designer. | Manual |

## Tag Structure in Ignition

The tag import files shown above are used to construct the tag hierarchy in Ignition that is shown below.

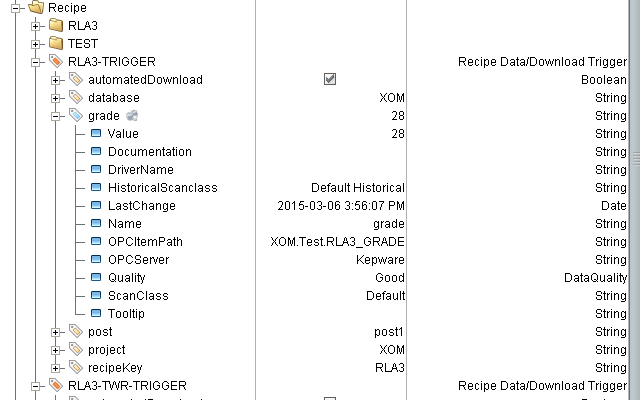


# Recipe Toolkit

The Recipe Toolkit consists of a SQL\*Server database schema, a DB Manager project, and a set of screens in the main Ignition project. The old architecture consisted of Microsoft Access database files, a Visual Basic DB Manager application and a module in G2 for downloading the recipe.

## Create Trigger for Automatic Download

A Download Trigger UDT needs to be manually configured for each console/unit where recipes need to be downloaded automatically.



# Diagnostic Toolkit

The Diagnostic Toolkit is a collection of applications designed for the detection, management, annunciation, and response to events. The output of any of the constituent tool is a diagnosis, a recommended response to the problem. Details of existing applications may be found in the references.

A custom module, Block Language Toolkit (BLT), has been coded by ILS Automation as a replacement platform for the Diagnostics applications. As part of the development effort, a series of migration tools has been developed.

The migration takes place in 3 steps:

1. Export from G2. This requires a new G2 module that traverses a diagram workspace and extracts information about its blocks and connections. The output is written as a JSON document.
2. Convert. Convert the JSON exported from G2 to JSON compatible with the new BLT platform.
3. Import into Ignition. This step makes use of the standard BLT import mechanism for diagrams. This is a manual operation. An entire Application may be imported in a single step.
4. Tag Mapping. Step #2 utilizes a tag translation database table that serves to translate from G2 names to Ignition names.
5. Conversion of Calculation Methods. This step makes use of the standard BLT import mechanism for diagrams. This is a manual operation. An entire Application may be imported in a single step.

## Export from G2

This requires a new G2 module that traverses a diagram workspace and extracts information about its blocks and connections. The output is written as a JSON document.

For each block the following is recorded:

1. Block class
2. Block unique ID
3. Block position – x,y in workspace coordinates
4. Block attributes – name-value pairs, these can be nested

For each connection:

1. Upstream block unique ID
2. Upstream block connection port name
3. Downstream block unique ID
4. Downstream block connection port name
5. Connection class

From a high level, the format of the JSON output is:

{“blocks”: [ { *serialized\_block1* }, …],

“connections”: [ { *serialized\_connection1* }, ….] }

As in Python syntax, {} denotes a dictionary, [] denotes a list. Whitespace is ignored. The details of the serialized blocks and connection are flexible.

During development, JSON syntax can be validated via the following free website: <http://www.freeformatter.com/json-validator.html>.

## Tag Mapping

Configuring tag translation is a manual task that requires a deep understanding the existing G2 application. While G2 has multiple mechanisms of referring to specific object instances, Ignition is much more limited – a named object is either represented by a tag (or UDT), or a database entry.

See section 2.6 above for a list of types of entities that must be mapped. The process of mapping tags is tedious and time consuming. Moreover there are two outputs. The first output is an import file (described above) which generates tags in Ignition. The second output is a SQL file that creates the mapping tables used for both block conversions and procedure generation. It is imperative that these two operations remain in sync. The file with SQL inserts will be made available to the migrators directly.

## Convert

The JSON document exported from G2 contains only references to G2 entities. It has no toolkit-specific information.

Conversion to BLT-specific JSON is performed by a *blt\_migrate* tool. The tool accepts a G2-JSON document on its standard input and writes BLT-JSON on its standard output. (This is done to facilitate bulk processing with scripts). The *blt\_migrate* tool connects to a database that contains translation tables mapping such things as G2 class names to BLT class names, and G2 method names to BLT procedures.

The migration tool, *blt\_migrate*, is a stand-alone Java application. It can be run on any platform and does not necessarily have to be run on a production system. The application basically accepts a file that represents an exported application and generates a corresponding file that is directly importable into Ignition. It is only this output file that must be transferred to the production environment.

The application accesses a SQLite database that contains mapping tables that translate between G2 names and Ignition names. The translations are:

* G2 class names -> Ignition block class names
* G2 item names -> Ignition tag paths
* G2 property names -> Ignition block properties
* G2 callback names -> Ignition python paths

## Calculation Procedure Translation

An automated procedure translation tool performs translation of the calculation callback procedures into Python. The automatic calculation is comprehensive except for the following G2 constructs:

* Do in parallel
* Go to
* Iterate over all instances of a class

The translations make extensive us of the translation tables, and so must be re-run whenever entries change.

Exxon has requested that the calculation methods reside in *project* scope Python. At the present time this requires the manual step of cutting generated code from the git translation output repository and pasting it into the Ignition project scope Python editor.

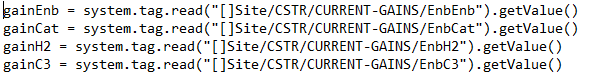
Any methods/procedures referenced from within the top-level calculation method must be hand-coded (and must support the same arguments and return values as the original G2 version). Any procedure name changes must be fed back into the translation tables for a re-run of the automated translation.

Gains:

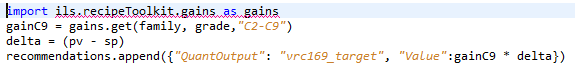
G2:



Raw Migration:



Final Migration:



Note: parameter for gain must be capitalized

## Import into Ignition

This step makes use of the standard BLT import mechanism for diagrams. This is a manual operation. An entire Application may be imported in a single step.

# Sequential Control

The Sequential Control Toolkit migration follows a similar workflow as the diagnostic toolkit migration.

1. Export from G2
2. Convert export file from G2 format to Ignition format
3. Import into Ignition

## Export from G2

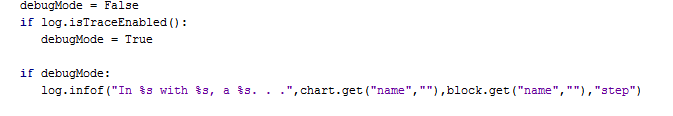
The ILS-Exporter module contains utilities to export S88 recipes.

### Callback Export

Callback blocks contain a reference to a callback procedure. Each callback procedure will be exported to a text file.

Callbacks written in G2 must be manually translated to python. Several aspects of the G2 callbacks are not needed in the python and can be eliminated. This includes:

1. any line referencing “debugMode”



1. “block state”



Other lines require minor changes:

1. “log.infof(…)” changes to “log.tracef(…)”



converted to:



1. Scope: G2 used the function “S88Scope” and required the use of strings. The new system imports constants to be used instead.

Raw Migration:

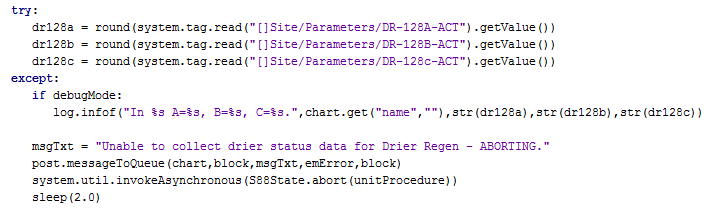


Final Migration:

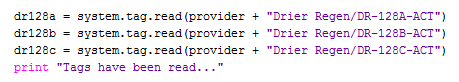


1. “try-catch” when reading from tags is unnecessary since a system.tag.read cannot fail.

Raw Migration:



Final Migration:



1. A new function was written for posting messages to the queue

Raw Migration:



Final Migration:



1. Change to function call used to get data:

Raw Migration:



Final Migration:



1. Sending OC Alerts

Raw Migration:



Final Migration:



1. Getting Loggers

Raw Migration:



Final Migration:



## Convert

The export from G2 is an XML text file. This step converts the G2 specific data in the export to Ignition specific data.

### Procedure Conversions

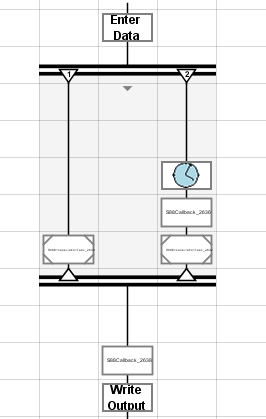
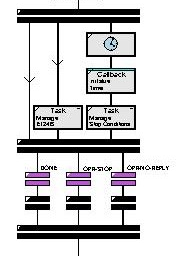
The SFC step “onstart” scripts are automatically generated subject to the same restrictions as described for the diagnostic toolkit. The translations must be made before conversion of the charts since the Python becomes embedded as part of the project.

As with the Diagnostic Toolkit, any second-tier method references must be hand-coded. Similar warnings are in effect regarding the need to synchronize the translation mapping tables with actual tag paths and procedure names.

### Block Conversions

#### Parallel Transitions

In the old platform, the top transition and the bottom transition were two separate blocks and the relationship between them was not explicit. These will be exported in the XML file as two separate blocks. In Ignition, the parallel transition is a single block. Additionally, it is sort of a container where blocks are placed inside of it. During the conversion the two old parallel transitions will bee converted to one new one.



In the above case, the original SFC contained an odd number of parallel blocks which is impossible in the new system. In addition, the straight connection between parallel blocks in the original system is now unnecessary. Finally, the termination transition blocks (purple transitions) present in the old system do not exist in the new. Instead, they were implemented into the cancel conditions of the parallel block.

### Recipe Data Conversions

#### Simple Data Sequence Type

The recipe data type in Ignition corresponding to a S88-Simple-Data is simply value. The class in G2 supported a simple type of sequence. This will not be supported in the Ignition Value class. Therefore this will be converted to an Ignition List data type.

## Import into Ignition

Once chart conversions are complete, an automated process converts the chart with all its steps and embedded Python into an Ignition partial project. When importing this file into Ignition, the new project will replace any existing project of the same name.

### Callback Procedures

There are several special considerations for callback procedures.

#### Python Location

There is a separate utility that will convert the G2 callbacks to Python. Ignition allows the Python for an action step to be in one of three places:

1. Directly in the step
2. In the Shared scope resource.
3. In External Python.

In the latter two strategies the action step would reference the procedure with the appropriate scope reference. The automatic migration procedure will convert the G2 callback procedures to Python and embed them directly into the action step. If there are multiple steps calling the same callback, migration will place the converted callback procedure into each of the steps. It is up to the migrator to manually refactor the blocks and copy the common code to the shared Python scope (internal to Ignition Designer).

#### Long Running Callbacks

The old application supported two styles of callbacks: long running and short running. Long running callbacks have some sort of looping construct that generally has a sleep at the bottom of the loop followed by a check of some exit condition. Because the callback is long running it must have logic for listening and responding to commands to Abort, Stop, Pause, etc. The vast majority of callbacks are expected to be short running callbacks. It is anticipated that short running callbacks will be able to be translated with a high degree of accuracy. These will be called by the On Start handler in the action block.

Long running callbacks will need to be deconstructed and divided into two scripts, the first of which is called in the On Start handler and the other in the Timer handler.

# Lab Data Toolkit

The Lab Data toolkit is completely data driven. There are no site specific screens.

## Lab Data

1. Export the lab-data, lab-table, and unit parameter data from the old application. This will generate three XML files. Check the files into SVN in EMChemicals/Migration/*site*/Exports/LabData
2. Obtain the names of the HDA GSI interfaces that were used in the old system. Determine what the names will be in the new system and manually configure the interface in the gateway. Add the translation to the InterfaceTranslation table in the migration database.
3. Using the Migration/Lab Data window in the Ignition Designer:
   1. Load the lab-data file
   2. Load the lab-table file.
   3. Load the unit parameter file
4. The data that is inserted into the database will be included in the database create script. The tags that are built will be included in the tag export that will be imported at the site.

## Lab Feedback Control

Lab-Feedback-Control is configured by TODO

1. **Diagnostic Toolkit Migration on Windows**
   1. **Cygwin**

The migration scripts are currently written as *bash* shell scripts. To run these on Windows requires *cygwin*. *cygwin* is a freely available collection of POSIX tools at <https://www.cygwin.com>. When downloading the shell, also include SQLite.

The cygwin packages which should be installed in addition to defaults, include:

* base: bash, gzip, grep
* database: sqlite3
* editors: nano, vim (or your favorite text editor)
  1. **SQLite**

SQLite was chosen because it is easy to script, has no license restrictions and no installation. There is also an easy-to-use GUI editor for Windows/Linux or OSX available at http://sourceforge.net/projects/sqlitedbrowser/files). In addition, databases can be edited with a Firefox add-in available at https://addons.mozilla.org/en-US/firefox/sqlite-manager. The database creation scripts can be edited with a simple text editor.

* 1. **Bash**

The scripts to create the database and process files are currently written in bash and checked into git. The scripts contain path references that are specific to the ILS Automation build layout, but can be easily modified for other environments.

The following is a list of relevant scripts:

* mkdatabase – constructs the translation database from insert scripts in the mdb directory.
* copy-json – copies G2 export files from the subversion repository into a migration work area
* migrate – executes the blt-application jar file.
* mktags – reads the translation database and generates a list of tags referenced by the migrated application.

These scripts are currently under version control in the ILS Automation git repository under blt/migration.

1. **S88 Migration on Windows**

This appendix describes specific migration steps on Windows

* 1. **Cygwin**

See section A.1

* 1. **SQLite**

See section A.2