Ignition™ Applications
Tool-chain Installation Manual

ExxonMobil Chemical Company

Document Version: 1.2

Dec 12, 2017

1. TABLE OF CONTENTS

2. CHANGE HISTORY

Version	Date	Author	Changes
0.1	Oct 1,2016	C. Coughlin (ILS)	Outline
1.0	Dec 1,2016	C. Coughlin (ILS)	Initial version
1.1	Dec 4, 2016	C. Coughlin (ILS)	Compressed figures (slightly)

3. INTRODUCTION

This document contains installation and configuration instructions for the tool-chain needed to generate ExxonMobil Chemical Company toolkit applications from source. The applications in question are those that are hosted on the Ignition™ platform from Inductive Automation provided through ILS Automation.

Some components of the applications are already delivered in source form, namely Ignition projects, Python scripts and various configuration files. These components are not addressed in this document. Instead, the focus here concerns the Ignition extension modules which are written in Java.

The culmination of the build-from-source addressed in this document is creation of an installer to load the newly created modules.

4. TOOLS INSTALLATION

One of the important features of an Ignition application is platform-independence. Likewise, the tools for building these applications are not tied to a particular operating system. However, the descriptions and configuration instructions below are all tailored to a Windows platform.

4.1. Java

The first step in preparing the build environment for the ExxonMobil Chemicals Ignition Applications requires installation of Java 8. It is important that tis be the (Java Development Kit) JDK rather than the Java Runtime Environment (JRE) version. See http://www.oracle.com/technetwork/java/javase/downloads. Choose the latest Windows 64bit version. At the time of this writing, the version is 8u151. DO NOT use Java 9 (yet).

Download and run the installer executable. Install the “Development tools” into the default location (e.g. *C:\Program Files\Java\jre1.8.0_u112*). On completion you will notice a companion directory (e.g. *C:\Program Files\Java\jdk1.8.0_u151*) that has a *bin* subdirectory with the Java compiler (*javac*) and other development tools.

4.2. Ignition

The development system should have its own installation of Ignition. The application is available from the Inductive Automation website at http://inductiveautomation.com/downloads/ignition. We recommend that the installation be into a generic directory – one that does not include Ignition version (e.g *C:\Applications\ignition*). This practice allows future updates to be installed without changing the directory name or variables that point to it.

It is important that the *lib* and *user-lib* subdirectories of the Ignition installation (and their children) are writable by the user running the build scripts. You may have to manually configure this after each Ignition upgrade.

4.3. eclipse

eclipse is an open-source Integrated Development Environment (IDE) for both Java and Python. The available eclipse versions are listed at: http://www.eclipse.org/downloads/packages. At the time of this writing, the latest version is "Oxygen". Installation details that follow will refer to this version. Differences with future versions can be expected to be minor.

Download the package "Eclipse IDE for Java Developers" – Windows 64 bit. Unzip the installation bundle and move the *eclipse* subdirectory into a directory for applications, say *C:\Applications\eclipse*. Create a shortcut to the *eclipse* executable inside that folder. Drag the shortcut to your desktop. Pin to the taskbar, if so desired.

NOTE: We have run into permissions issues if we install the *eclipse* application into the standard `Windows C:\\Program Files` area.

Thinking ahead, create a directory to receive the source distribution from ILS Automation, say *C:\repository*. Make a sub-directory under that, *git.* We will use this as our initial workspace while we complete the eclipse configuration.

Start *eclipse* and point it to our initial workspace.

4.3.1. PyDev

PyDev is an *eclipse* plugin for development of Python code. Under the *eclipse* Help->Install New Software menu, add a new update source:

Name: *PyDev*
Location: *http://pydev.org/updates*

Before updating, define a HOME environment variable. In the File Browser, right-click on "This PC" and select *properties->Advanced System Settings->Environment Variables*. Set the value with a path to your home directory, e.g. *C:\Uers\chuckc*. Restart *eclipse*.

Select the PyDev updates. Restart *eclipse* as directed.

4.3.2. HTML/XML editing

The *ant* build scripts and several of the configuration and license files are HTML or XML format. Eclipse already includes rudimentary editors for these files. If so desired, these can be enhanced by installing the Eclipse Web Development Toolkit.

4.3.3. Jython

jython is Python interpreter required for integration with Ignition. Version 2.5.3 is required. *Jython* may be downloaded from http://www.jython.org/downloads.html. Run the installer and store the result in some convenient directory, e.g. *c:\Applications\jython-2.5.3*.

In *eclipse*, *Window->Preferences->PyDev->Interpreters->Jython Interpreter*, choose *New*. Name it `Jython2.5.3`, and browse to the executable which is *jython.jar* in the *Jython* directory we just made. Select all the packages and *Apply*.

4.3.4. Java

We've already installed the JDK onto our system, but it is very likely that the *eclipse* Java environment is pointing to the JRE. We can rectify this by configuring the default Java environment to be the JDK. In eclipse, Windows->Preferences->Java->InstalledJREs, add the path to the JDK and make it the default (if this is not already the case).

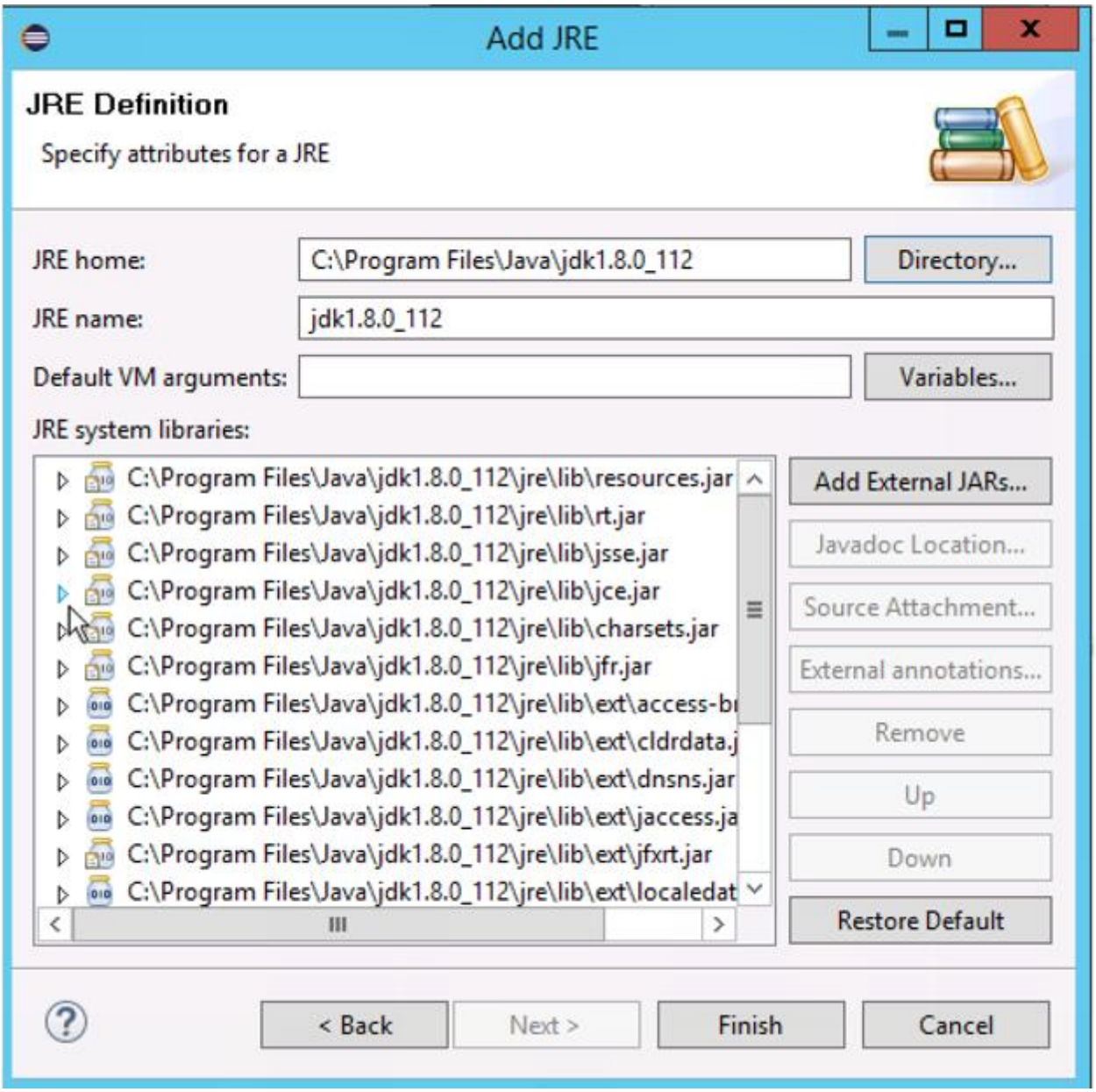

Figure 1: Defining the JDK as a Java Runtime

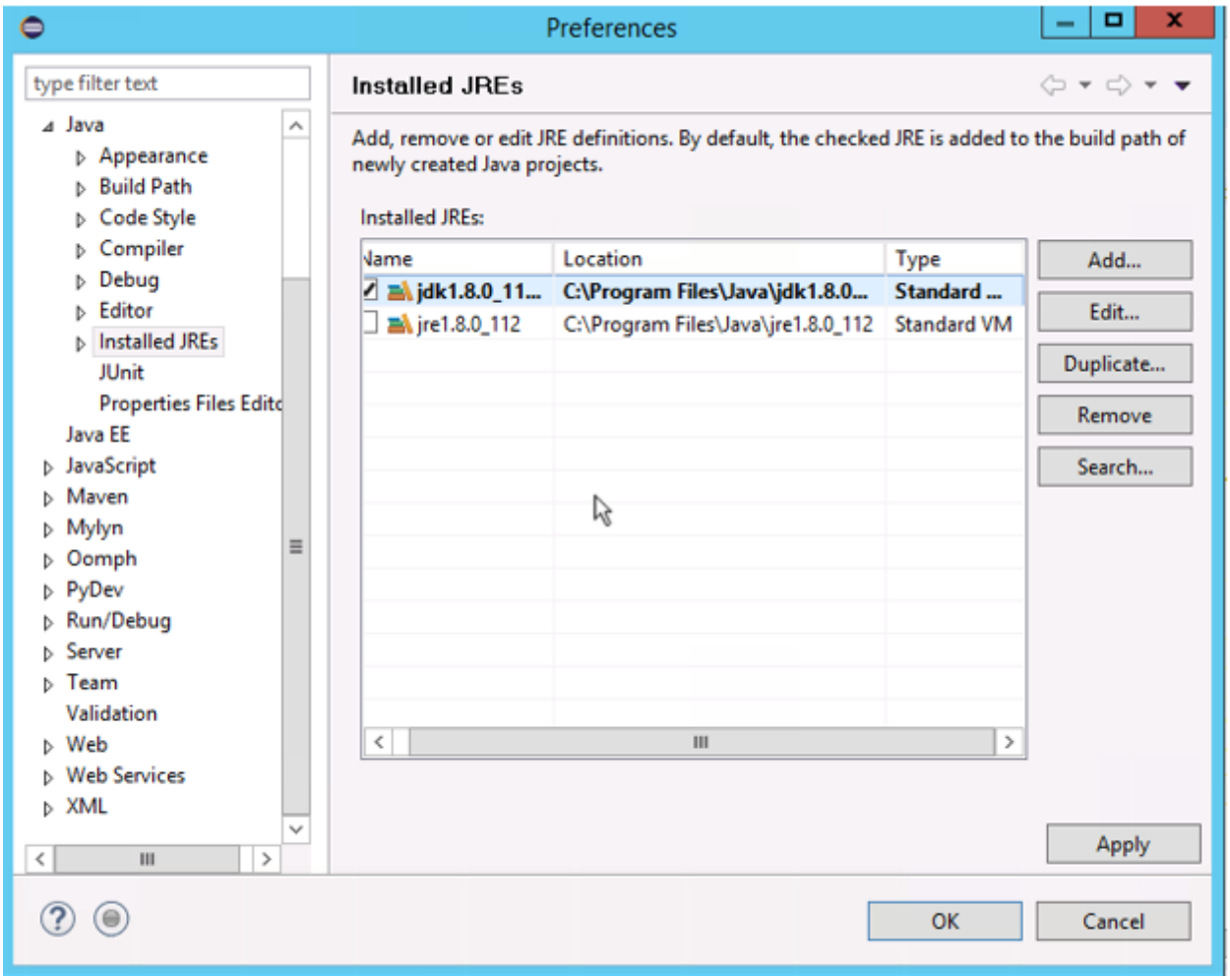

Figure 2: Setting the JDK as the Default Runtime

Additionally, set the operating system environment variable `JAVA_HOME` as the path to the JDK.

4.4. Module Signing

In order to load a module into Ignition (without a Developer license), the module must be signed. The signing is handled directly by the build scripts using the Java application *module-signer* from Inductive Automation. We've distributed the source code in `tools/module-signer`. It can be built, if necessary using Maven (an open-source build tool, but not included here).

The module signer requires keys and a signing-certificate. These can also be generated via open-source tools – but we have provided pre-built artifacts in our distribution. They may be found in `tools/keys`.

5. ENVIRONMENT VARIABLES

In order for the build scripts to function in a variety of environments, the system layout is described in terms of environment variables. As we've pointed out earlier, environment variable definitions are accessed from the Windows file browser under "This PC", selecting *properties->Advanced System Settings->Environment Variables*.

5.1. IGNITION_HOME/IGNITION_PYLIB

Set the variable `IGNITION_HOME` to the root of the Ignition installation directory, e.g. *C:\ignition*. A companion variable `IGNITION_PYLIB` should be set to the *user-lib/pylib* subdirectory under `IGNITION_HOME`, e.g. *C:\ignition\user-lib\pylib*.

5.2. GIT_REPO

Set the variable `GIT_REPO` to point to the *git* sub-directory in our source repository area, e.g. *C:\repository\git.*

5.3. SVN_REPO

Set the variable `SVN_REPO` to a *svn* sub-directory of the source repository, e.g. *C:\repository\svn.* It is important that this be a sibling directory of the *git* area. This directory is used only when building installers.

6. DELIVERY BUNDLE

The source code is delivered in a single zip file. If replacing an existing source distribution, the old source should be removed, before the bundle is unzipped. If ExxonMobil wishes to retain prior versions, then the prior as-delivered zip file should be archived.

6.1. Installing the Source Bundle

The source bundle is delivered as a single .zip file. It should be unzipped into the parent of the `${GIT_REPO}` and `${SVN_REPO}` directories. *Eclipse* must not be running during this operation.

Once the source bundle has been transferred to the development system, right-click on the zip file and select *Extract All...* Specify the repository as the target directory. The resulting directory layout is shown below:

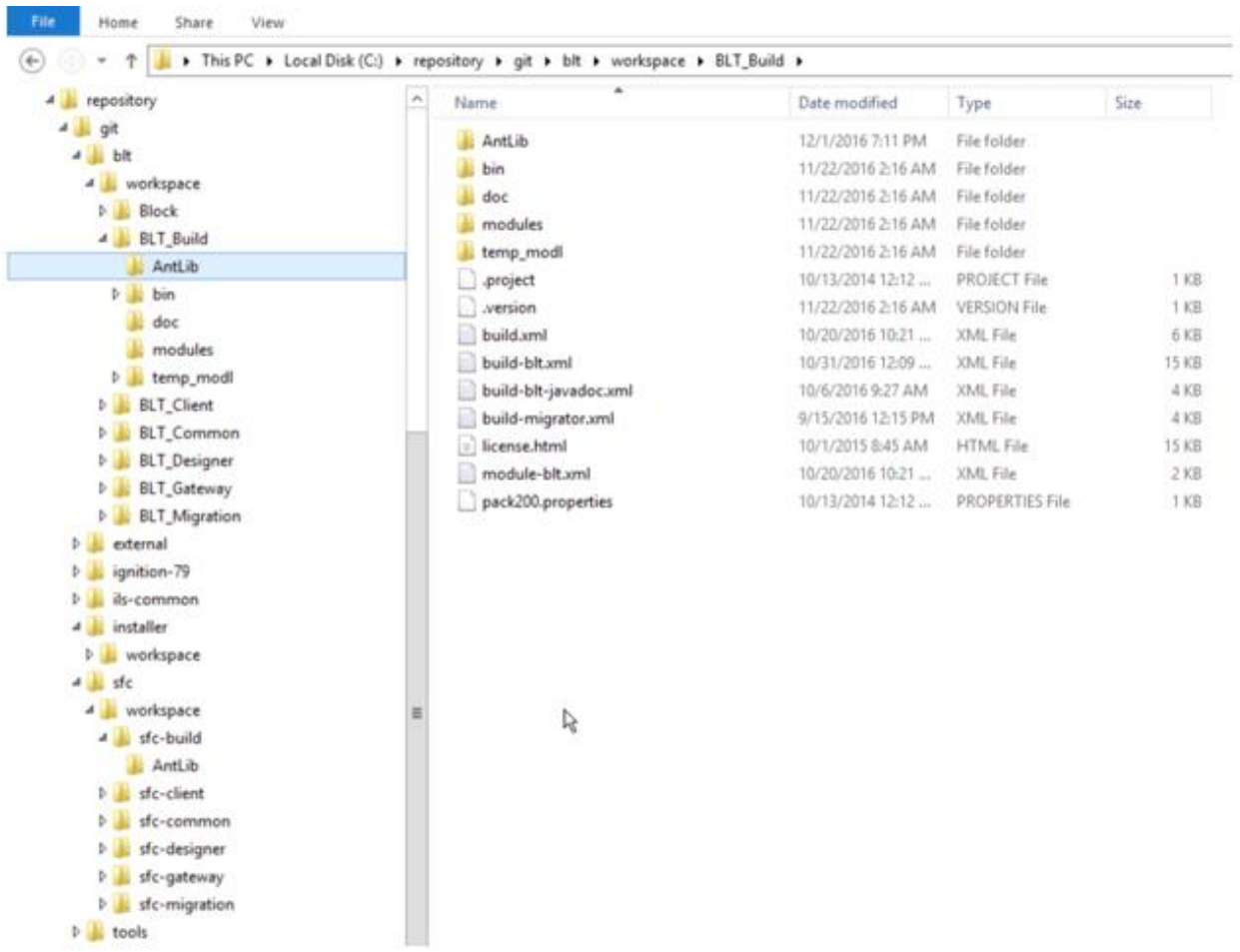

Figure 3: Directory Layout

This layout mirrors the ILS Automation development environment. It is important that it not be changed, as many of the *ant* build scripts rely on the fixed structure.

6.2. Loading into Eclipse

The source distribution contains *eclipse* projects. They may be imported directly into eclipse via the *File->Import->General->Existing Projects into Workspace* menu. Browse to the repository directory (the parent of `${GIT_REPO}` and `${SVN_REPO}`). Load all projects listed.

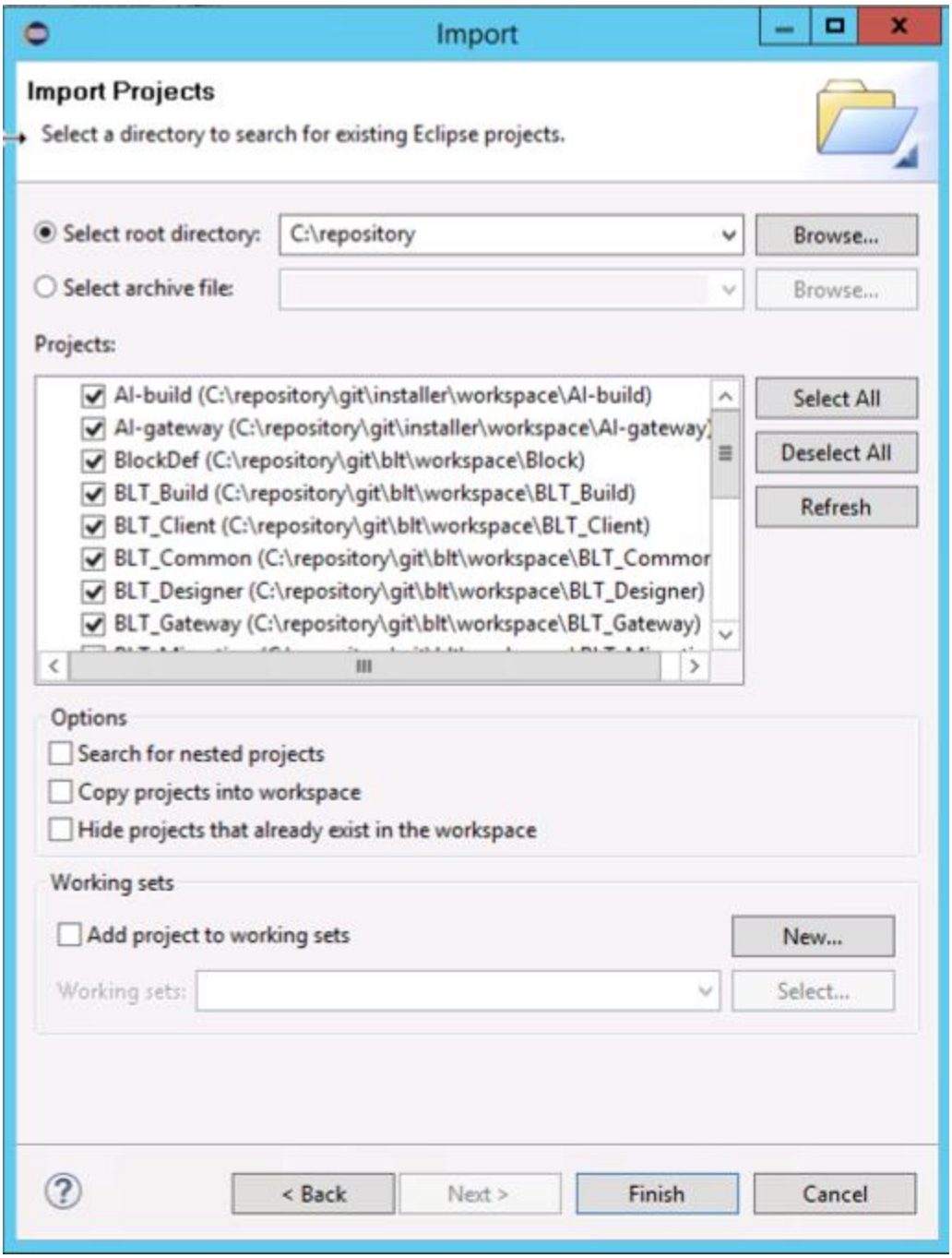

Figure 4: Eclipse Project Import List

Note that on subsequent loads of the source, the import step will be unnecessary. Simply refresh Eclipse using File->Refresh.

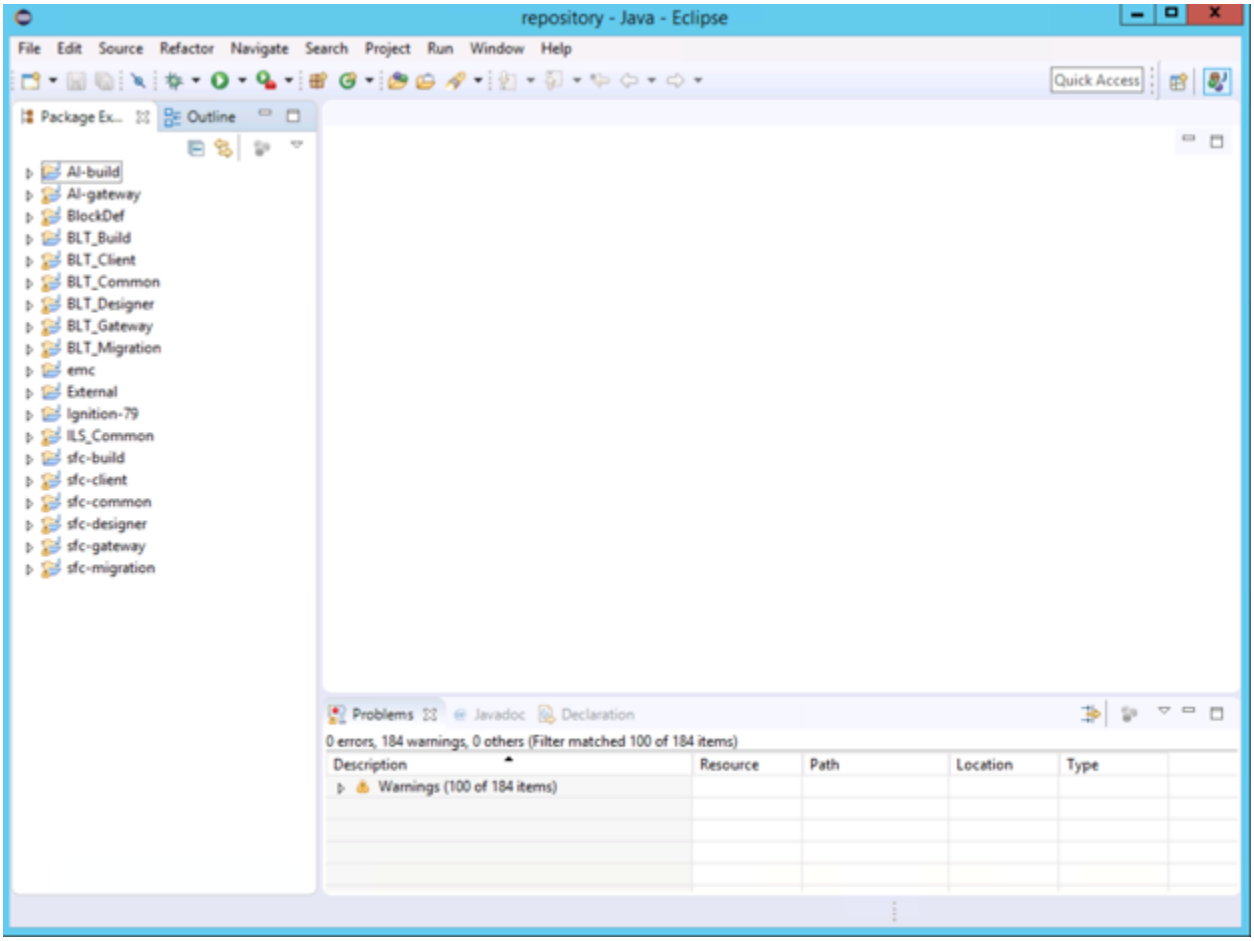

Figure 5: Projects Loaded into Eclipse

Note that the *TaskList* panel is not used and can be closed to create more space.

7. CREATING THE MODULES

The distribution contains source code for two Ignition modules:

- Block Language Toolkit – this module contains the code for the toolkit which supports the diagnostic toolkit. The module contains block definitions, graphics and the engine which runs.
- ILS-SFC – This is an extension of the Inductive Automation Sequential Function Chart package. The ILS extension provides support for custom blocks required by EM and the concept of recipe data.

7.1. Ant Script Configuration

ant is the tool used to script the builds. In general, the scripts contained in the distribution need to be configured in *eclipse* before they can be used. The configuration needs to be done only once and not repeated even for new source versions.

The individual configuration will be listed as each script is introduced below, but the general process is to right-click on the script from *eclipse* and choose *Properties*. Configure a new Run/Debug Setting as an *Ant* build. The argument settings are script-specific. However, make sure that the JRE setting points to the JDK, as shown below.

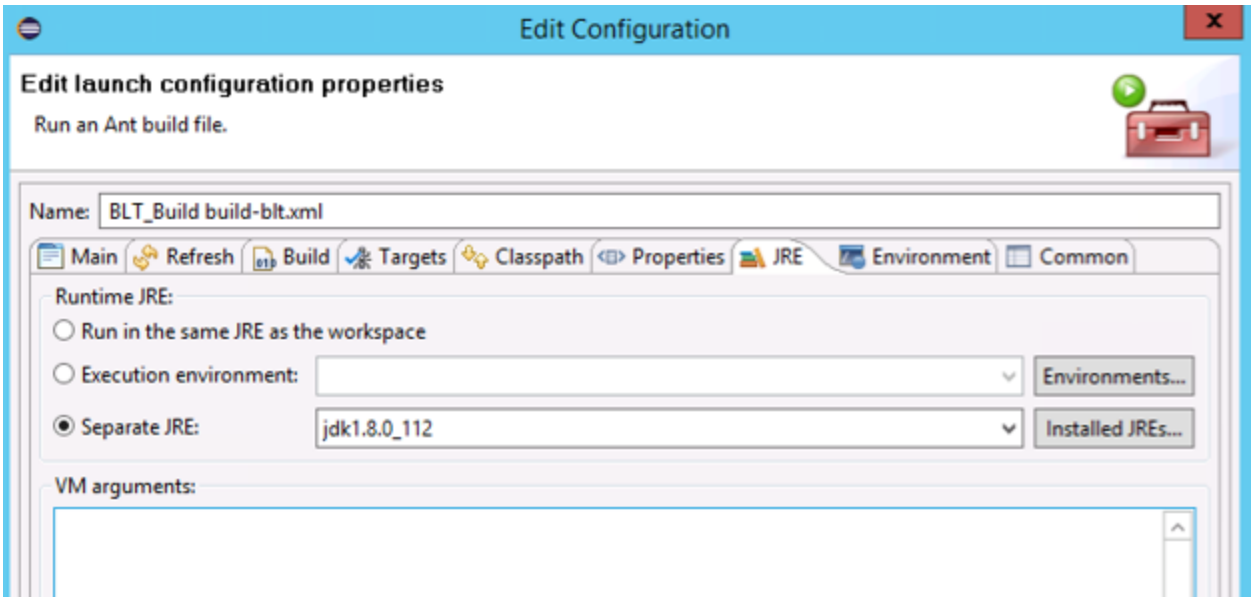

Figure 6: JRE Setting for Build Script

If the JDK was properly configured as the *eclipse* default, no change should be required. This is a critical setting, so is good to verify.

7.2. Building BLT

The figure below shows *eclipse* ready to build the BLT module.

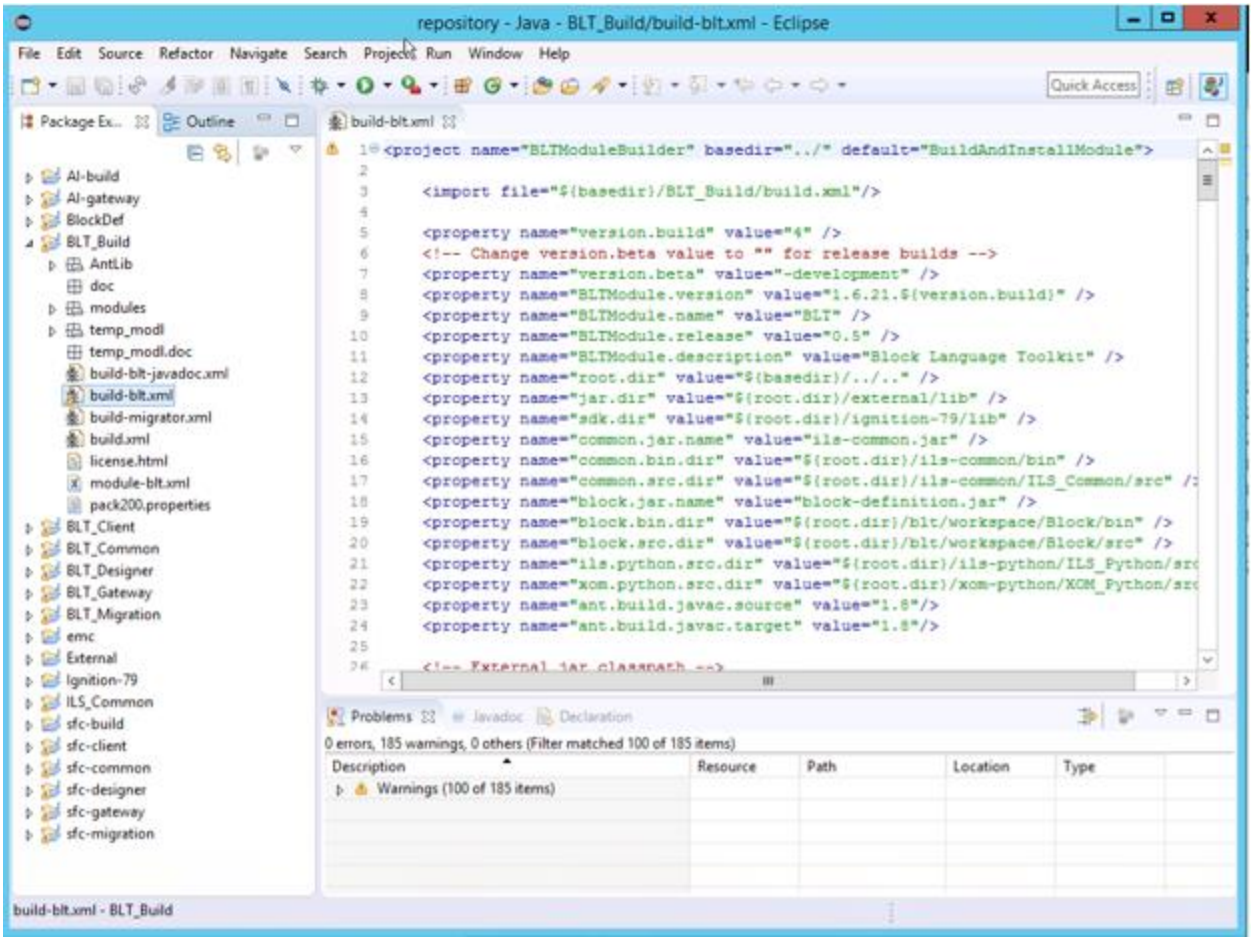

Figure 7: Build Area for BLT

The first script to run is: *build-blt-javadoc.xml*. Its purpose is to create the JavaDoc files that are accessible via the Ignition Gateway modules page. The script needs no further configuration. To run, right-click, then *Run As* an *Ant Build* (first option). You will notice a slew of warnings in the *eclipse* console. These refer to incomplete JavaDoc comments in the code. These are harmless and we are in the process of cleaning these up.

The second script to run is: *build-blt.xml*. Its purpose is to compile and build the BLT module. It requires a configuration as shown below.

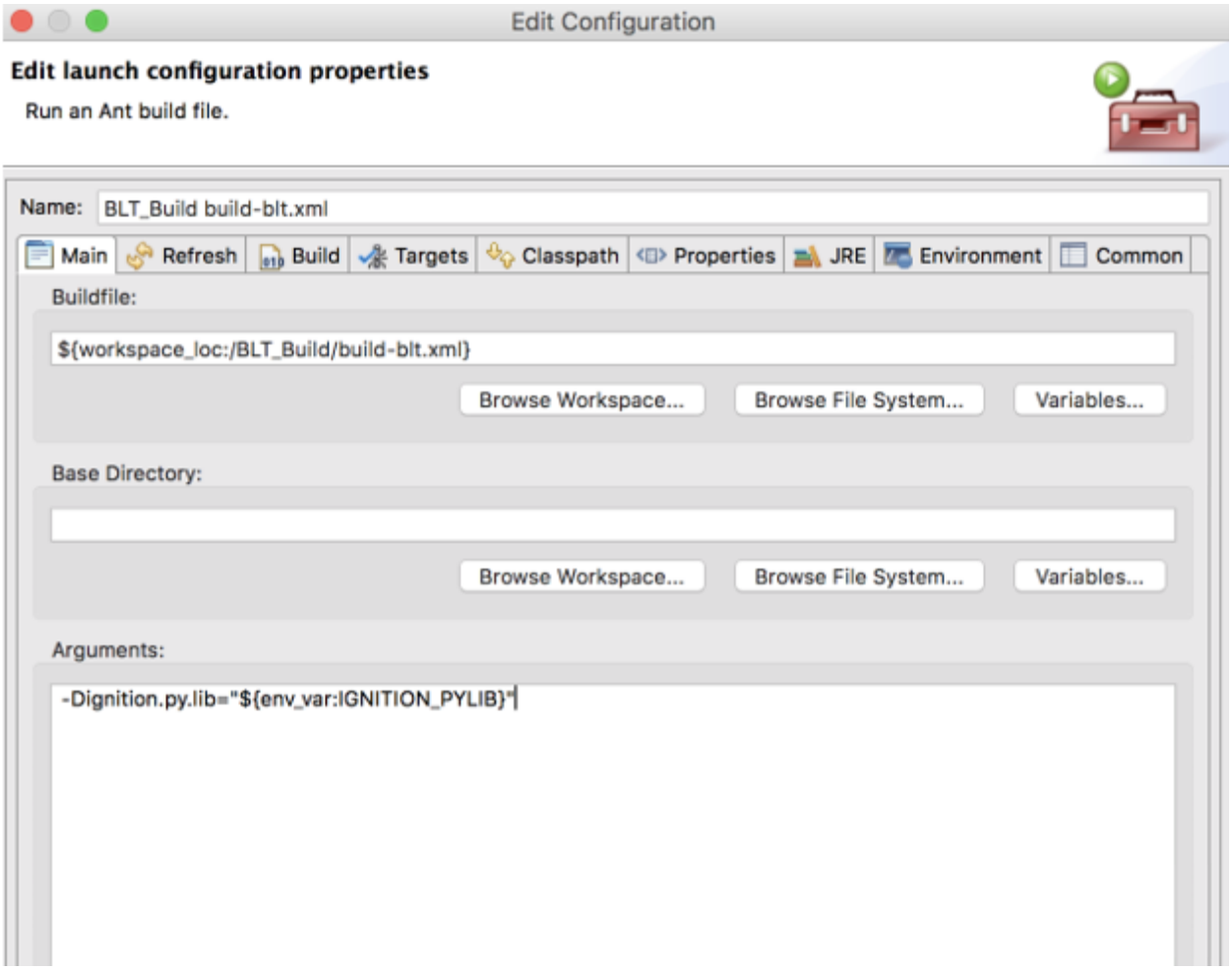

Figure 8: build-blt.xml Configuration

This script is designed to install the newly built and signed module into the local gateway, as long as it Is running. It also installs jar files that are used by external Python. A Gateway restart is needed for these to take effect.

7.3. Building ILS-SFC

The figure below shows *eclipse* ready to build the ILS-SFC module.

.

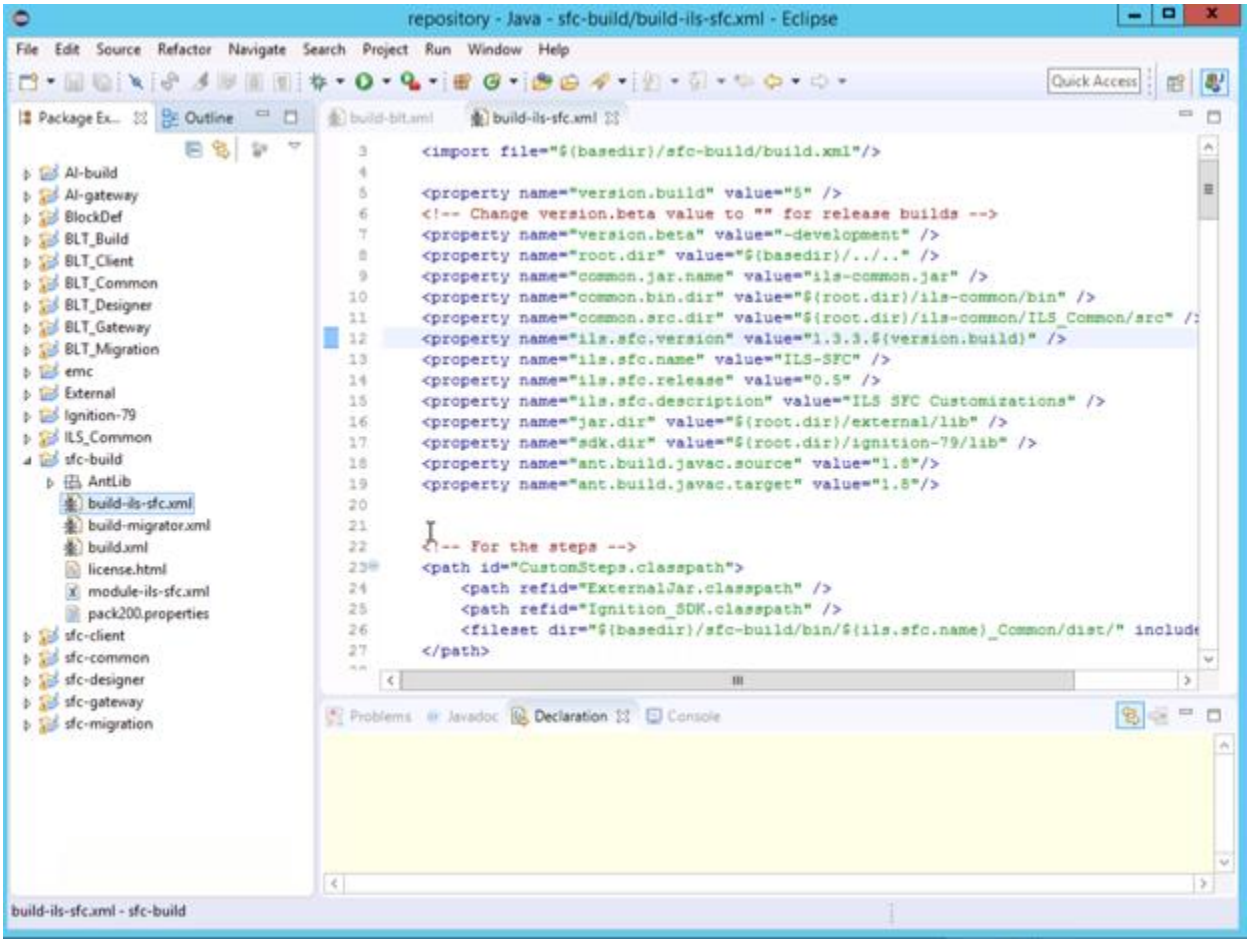

Figure 9: Build Area for ILS-SFC

The build script is: *build-ils-sfc.xml*. Its purpose is to compile and build the ILS-SFC module. It requires the same configuration as the *build-blt.xml* script (See above).

As with the previous, this script is designed to install the newly built and signed module into the local gateway, as long as it Is running. It also installs jar files that are used by external Python. A Gateway restart is needed for these to take effect.

8. CREATING INSTALLERS

The distribution includes source code for the ILS Automation installer. Additionally, it contains configuration files which may be used to construct an installer that loads the modules built in the previous sections.

8.1. Building the Installer

The figure below shows *eclipse* ready to build the jar file that contains the Application Installer logic.

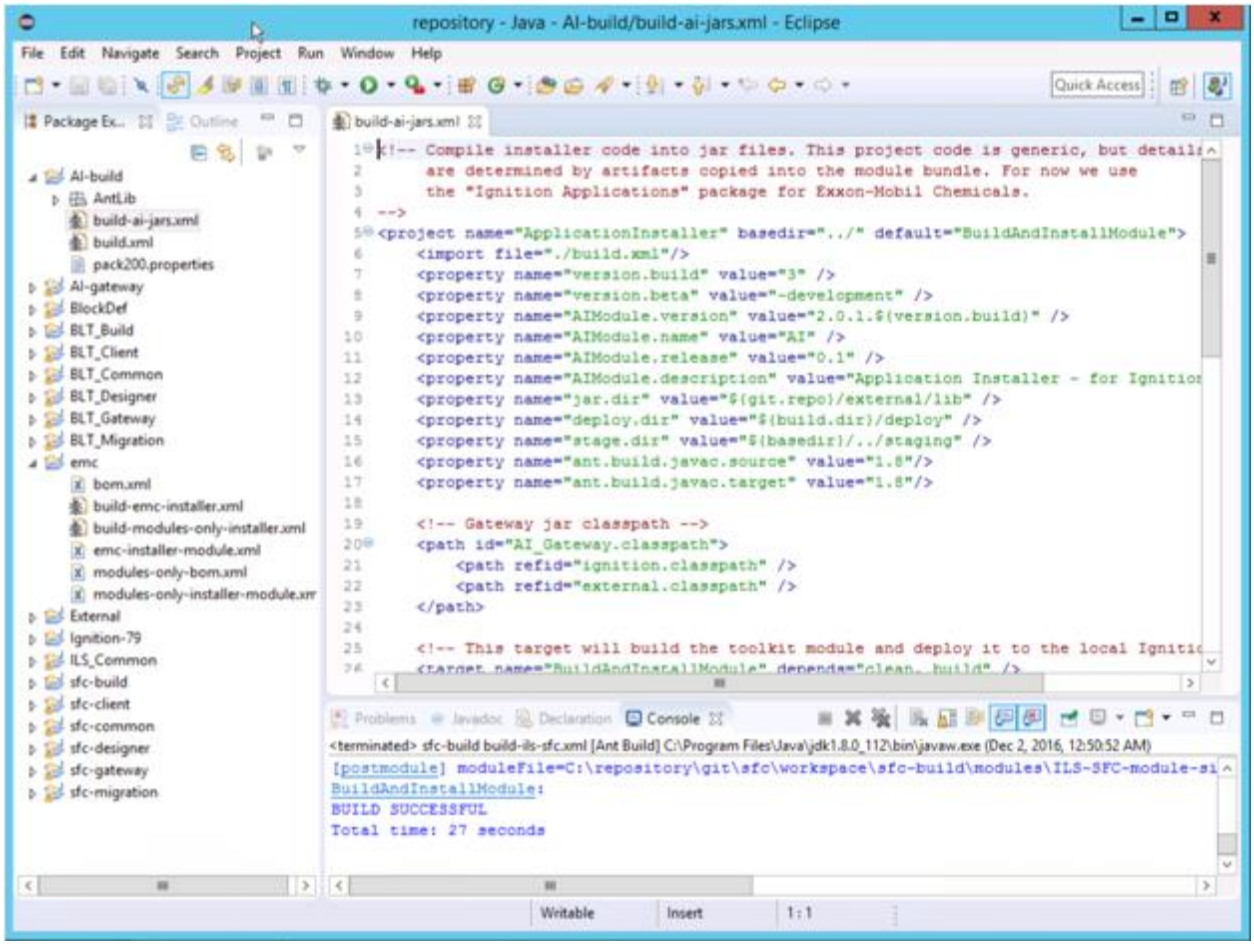

Figure 10: Build Area for the Application Installer

The build script is: *build-ai-jars.xml*. It requires the configuration shown below.

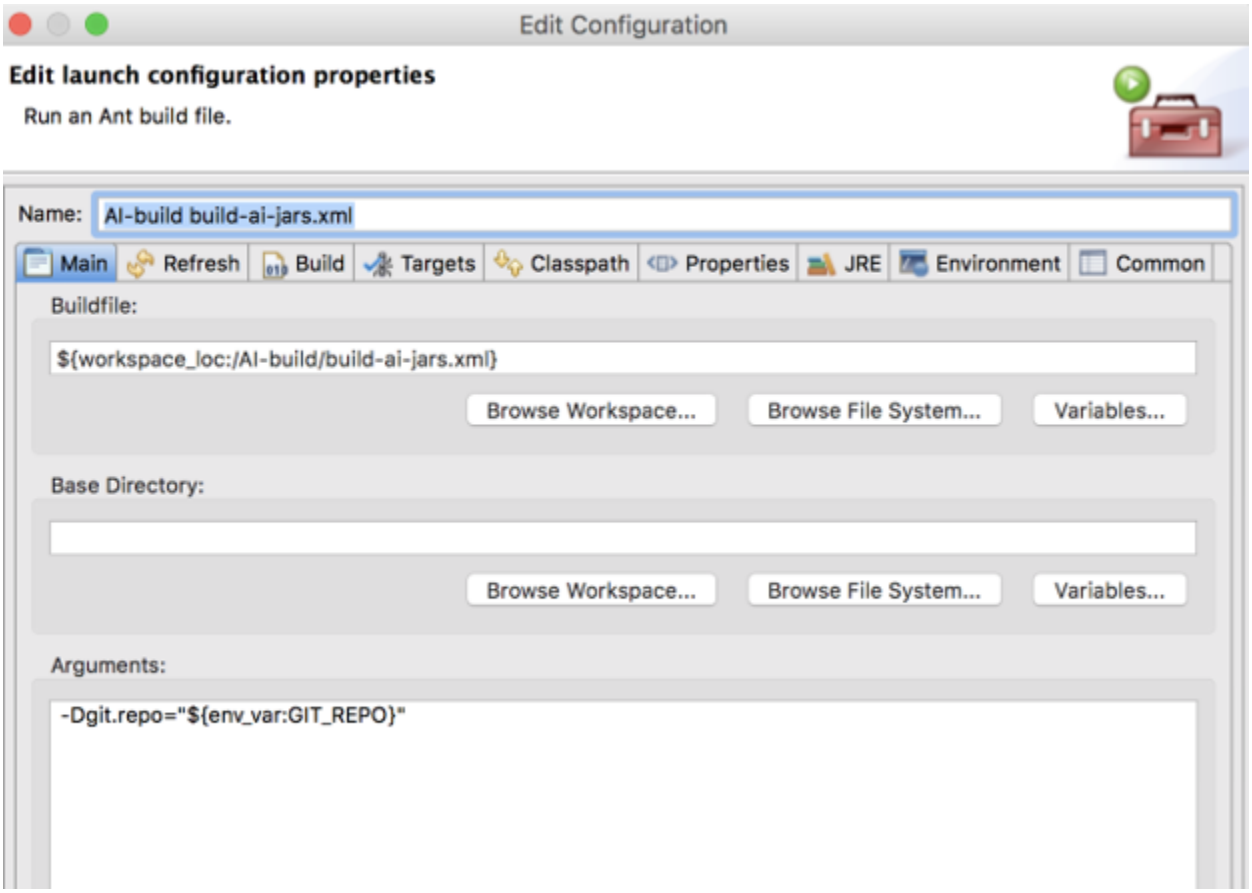

Figure 11: build-ai-jars.xml Configuration

Once the script has executed, the main logic for the installer is in place. The next steps involve configuration of installers that are specific to individual customers or situations.

8.2. Building a Module Installer

The source distribution contains configuration scripts for constructing an installer that simply updates an existing Ignition application with the newly built modules.

The script to run is in the *emc eclipse* project, named *build-modules-only-installer.xml.* It must be configured as shown below.

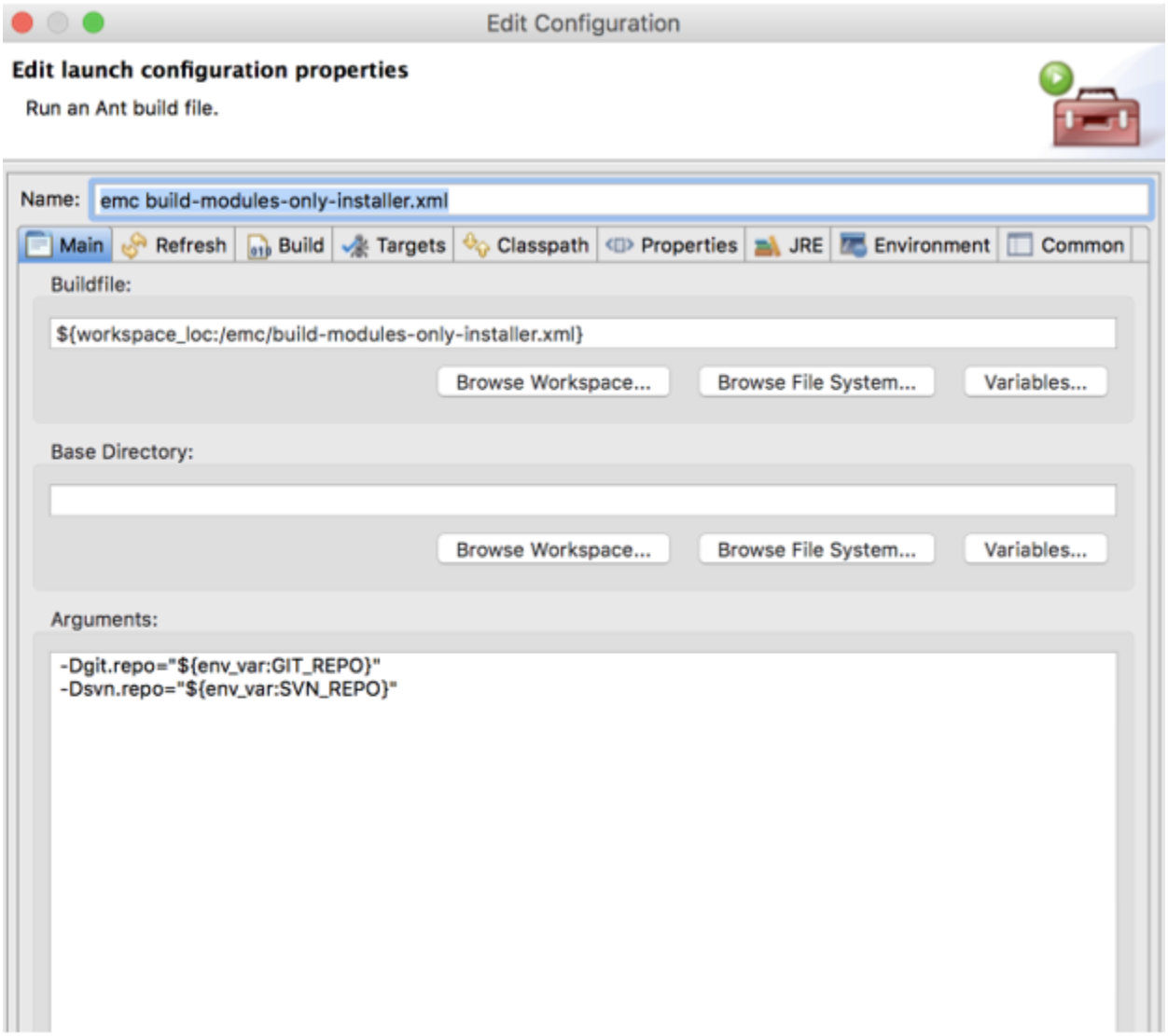

Figure 12: build-modules-only-installer.xml Configuration

This script relies on values contained in the DATE and VERS files found in the `${SVN_REPO}` area. These are text files which can be edited to assign a date and version to the installer being configured. The script uses build products of the BLT and ILS-SFC module compilation and copies them into the installation module.

On completion of the script, the installation module can be found in:

```
${GIT_REPO}\installer\workspace\emc\modules
```

Look for the signed module.

8.3. Configuring Other Installers

Building installers, in general, is beyond the scope of this document. A useful installer requires many artifacts beyond the two module files which we have built here.

For further instructions, see:

- Installer for Ignition-based Applications – distributed as part of the main release.
- The *emc* configuration scripts that were unused in the previous section are the scripts which are used to generate a full production release. They are good examples.

Note that an installer module is simply a .zip archive. The comprehensive production installer delivered by ILS Automation can be unbundled to reveal all of the various artifacts needed for a complete installation. These can be repackaged at will.