**Installer**

**for**

**Ignition-based Applications**

**Document Version:**

**Sept 20, 2017**

Table of Contents

Table of Contents

1. Introduction 2

2. Loading 3

3. Creating the Installer Module 4

4. Panels 5

4.1 Welcome Panel 5

4.2 License Panel 7

4.3 Backup Panel 8

4.4 Site Panel 9

4.5 Feature Panel 11

4.6 Interface Definition Panel 12

4.7 Cleanup Panel 13

4.8 Authentication Panel 14

4.9 Icon Panel 15

4.10 File Panel 16

4.11 Database Panel 18

4.12 Module Panel 20

4.13 Python Panel 20

4.14 ScanClass Panel 22

4.15 Tag Panel 23

4.16 Toolkit Panel 24

4.17 Project Panel 25

4.18 Documentation Panel 28

4.19 Source Panel 29

4.20 Summary Panel 30

4.21 Epilog 31

# Introduction

Ignition™ is an execution and development platform from Inductive Automation designed to support a wide-range of industrial control applications. While a basic Ignition-based application may require only a project file, a more comprehensive application may require a number of different components for correct execution. These components may involve a variety of file types and installation steps. The variety may be confusing to the end user. Absence of one or more of these components may yield an incomplete or inconsistent installation and result in subtle (or not so subtle) errors.

This document describes an ILS-Automation product that handles application releases in the familiar paradigm of an installer. Using the ILS Application Installer, the end user is presented with a wizard-style sequence of screens that handle installation of the various components that make up the target application. These components may be, among other things: full or partial projects, global projects, UDT definitions, icons, internal and/or external python packages, SQL update scripts, or Java-based modules.

With the ILS-Automation installer, the release bundle for a particular delivery is packaged into a single file, an Ignition module file. Embedded within the module are all the resources required for the target application. When the install module is loaded, an additional menu entry appears on the configuration panel in the Ignition Gateway. This entry launches a wizard-style sequence of panels that allow the end-user to complete the installation.

The screens with which the user interacts are completely configured via an XML file. This file describes the information on the screens as well as the locations of the actual resources within the bundle. This file is known as the bill-of-materials. The bulk of this document describes this file with sample output.

# Loading

The install bundle is loaded just like any other Ignition module - from the Gateway configuration page. It appears as “ApplicationInstaller” in the module list. This is the default name. It may be easily changed to something more appropriate to the product being installed by editing the application-installer-module.xml file in the module bundle as it is being created.



Figure – Gateway Modules View

Once the bundle is loaded, a new entry appears on the configuration panel. It names the product and provides a link to execute the installer.



Figure – Entry on Configuration Panel

The name that appears comes from the *<title>* element of the bill of materials.

# Creating the Installer Module

An Ignition module is simply a jar file. It can be created in a number of ways:

* manually – use the *jar* command that is part of any standard java distribution. E.g.

jar –cf <jar-file-name> <directory-to-be-jarred>

* shell script – create a *bash* script to collect artifacts and create the module.
* *ant* script – incorporate building an installer into your normal build process. An *ant* script may be executed directly from *Eclipse*, for example.

The standard components that make up the installer itself are required. These are:

* ai-gateway.jar – core installer code
* Iis-common.jar – common library for ILS applications
* module.xml – module descriptor. This file may be edited to change the module name and description as shown in the Ignition Designer.
* .application-installer – this is a zero-length file that is simply used to mark the fact that this is an installer module.

The files above are distributed by ILS Automation and constitute the core installer. These files must be stored in the root directory of the artifact bundle.

In addition to what is normal in a module file is an “artifacts” directory. This contains a bom.xml. (bill-of-materials) file, plus all of the resources that comprise the product.

In order that the module can be loaded into the Ignition Gateway without the end user having special permission, it must be signed. This is accomplished through the [Inductive Automation](http://www.inductiveautomation.com) website. You must be registered as an Ignition developer in order to perform this step.

# Panels

This section describes each panel type in detail showing both the XML fragments used for configuration and the resulting screen displayed to the user. Not every panel is applicable for every product. In some cases multiple versions of the same panel may be needed. This is all controlled by the makeup of the XML file, the product bill-of-materials.

Panel *type* is limited to a fixed vocabulary. Each type corresponds to a specific Java class that renders the page. Within a bill-of-materials, a page is given an arbitrary *subtype* that, in combination with the *type,* uniquely identifies each page.

## Welcome Panel

The initial panel displays a description of the product being installed and provides an opportunity to view release notes.

In the XML bill-of-materials document, an initial section of contains a list of properties that describe the product being installed. On startup, these are displayed on the Welcome panel along with corresponding values from the previous installation, if any. These properties are updated in the internal Gateway database once the installation is complete.

The properties shown in the XML fragment below should be included in every update. These are:

* product – the name of the product
* release – marketing name of this update
* date – release date
* version – an integer release number that increments with each update

These properties are defined in the bill-of-materials as shown below:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<installer>

<title>ACE Controls Installer</title>

<description>Installer for Ignition Application.</description>

<property name=*"product"*>ACE Controls</property>

<property name=*"release"*>1.2b5</property>

<property name=*"date"*>July 4, 2017</property>

<property name=*"version"*>2</property>

An arbitrary number of additional properties may be added to this list.

The welcome panel is defined in this XML fragment as:

<!-- The welcome panel must always be displayed, thus no version -->

<panel type=*"welcome"* essential=*"true"*>

<title>ILS Automation Installer for ACE Controls Applications</title>

<preamble>Welcome to the ILS Automation installer for the ACE

Controls Ignition Application. The table below compares

the existing installation of this product, if any, with

versions contained in this new installation. Before

continuing, please make sure that you have defined tag

providers and database

connections for both production and isolation

(test) environments. Also make sure that you have write

permission to the "lib" folder within the Ignition

installation directory.

</preamble>

<artifact name=*"notes"* type=*"pdf"*>

<location>artifacts/doc/ReleaseNotes.pdf</location>

</artifact>

</panel>

The *<artifact>* element defines where the resource is located within the installation module. The resulting screen is shown below:



Figure – Welcome Panel

The panel has a button for displaying release notes, and checkboxes that provide filters for subsequent panels to be displayed. If there is no release note artifact in the bundle, then the button will not appear.

The first checkbox option provides the ability to skip artifacts that have not been updated in the current distribution. The second choice allows skipping installation of artifacts that are not necessary for production. These may include: example projects, documentation, test scripts or source code.

## License Panel

The license panel provides a means to view and accept the End User License Agreement associated with the product to be installed. If the current version of the agreement has not been accepted previously, then the *next* button is disabled preventing further navigation until the acceptance is granted.

The license panel is defined as:

<panel type=*"license"* essential=*"true"* version=*"1"*>

<title>End User License Agreement (EULA)</title>

<preamble>Click on the button below to view the license that

specifies permissible use of the software to be installed. Select the checkbox to indicate acceptance of the terms.

You must accept the license terms to proceed.

</preamble>

<artifact name=*"license"* type=*"html"*>

<location>artifacts/license/license.html</location>

</artifact>

</panel>

As before, the *<artifact>* element defines where the resource (in this case the license file) is located within the installation module.

 Figure – License Panel

The panel has a button for displaying the license agreement.

## Backup Panel

The backup panel is an optional panel that allows the user to make an Ignition backup before proceeding further. It should be noted that an Ignition backup does NOT include modules, nor does it backup database instances. Thus it does NOT provide a straightforward way of reversing the install.

Note that the backup panel is configured as non-essential and has no version. It does NOT indicate completion.

<panel type=*"backup"* essential=*"false"*>

<title>Archive into an Ignition Backup</title>

<preamble>This is an optional step. Once created, a backup

allows you to revert the previous state of the application.

</preamble>

</panel>

 Figure – Backup Panel

Once the user selects the button to “Perform Backup”, the browser will (depending on how it is configured) display a dialog that allows selection of the destination for the resulting *.gwbk* file.

## Site Panel

The site panel handles cases where either:

1. The installation package is built with options that are specific to several locations. For example, the installer may contain tag, project, database or other artifacts that are specific to different sites. When the installer executes and the site is identified, it handles installation of the correct artifacts for that particular site.
2. The installation package is built to install several identical projects onto the same gateway. While identical, each project is assigned a different name and will be associated with different tag provider and/or database instances.

Not all panels necessarily apply to all sites. A “site” property specified on a panel definition may be used to restrict the panel to a specific site. If no “site” property is present, then the panel will be used for all sites. Multiple “site” property definitions are allowed.

The site panel requires metadata that describe the artifacts and their handling that are specific to each site.

Metadata definitions are part of the panel XML as described below. These definitions consist of a list of *site* elements each containing artifacts of various types that are specific to the named site.

Each *site* element contains a list of artifacts for that site, if any. Potential artifact types are:

* project – the name of a project required at this site. This name is overrides the name found in the *project* panel. Note that the same artifact location may be used for multiple projects.

If required, *tag*, *database* or *profile* properties may be nested inside the project property if the site-wide values are not appropriate.

* database – the name of a datasource to be assigned to the site. Optionally it may be specified as the default production or test datasource. The set of datasources required at any particular site must be pre-defined in its Gateway before each installation. Unlike with a single-site installation, the datasource names are pre-set in the installer.
* tag – the name of a tag provider to be used at the site. Optionally it may be specified as the default production or isolation mode provider for the site. The set of providers required at any particular site must be pre-defined in the Gateway before each installation. Unlike with a single-site installation, the provider names are pre-set in the installer.
* properties – specific properties may be assigned to the site. These are the same properties available with the definition panel:

<property name=*"database"* type=*"production"*>DB</property>

<property name=*"database"* type=*"isolation"*>TestDB</property>

<property name=*"provider"* type=*"production"*>Tags</property>

<property name=*"provider"* type=*"isolation"*>TestTags</property>

Since property values are specified, multiple entries may be present in each category.

*project*, *database* and *tag* panels coordinate with a site selection. Once a site has been selected, these panels will automatically install the proper set of artifacts associated with that site. See the sections dealing with these panels for more information.

In general, panels that are configured without a *site* property will be applied to all sites.

Figure – Site Panel

Depending on the presence or absence of database and tag properties, this panel may replace the *interface definition* panel for multi-site installers.

## Feature Panel

The feature panel allows the following panels of the installer to be dependent on whether or not a specified feature should be installed. The page contains of a list of check-boxes, one for each feature. The features correspond to artifact elements in the bill of materials. The following artifact properties define the sub-panels:

* name – the name of the artifact is displayed as the sub-panel title. It is the name of the feature.
* comment – the contents of this nested element is displayed as the text next to the checkbox.
* script – an optional nested element with a complete python path/module name. The module is expected to accept a single, a Boolean flag describing whether or not the feature is present. The intent of this entry is to allow the application logic to react to the presence or absence of a particular feature.

Not all panels are necessarily applicable to all features. A “feature” property specified on a panel definition may be used to restrict the panel to a specific feature. If no “feature” property is present, then the panel will be used for all features. Multiple “feature” property definitions are allowed. The value of the property is a string TRUE or FALSE signifying whether the panel should be applied in the presence or absence of the feature, respectively.

In general, panels that are configured without a *feature* property will be applied generically.

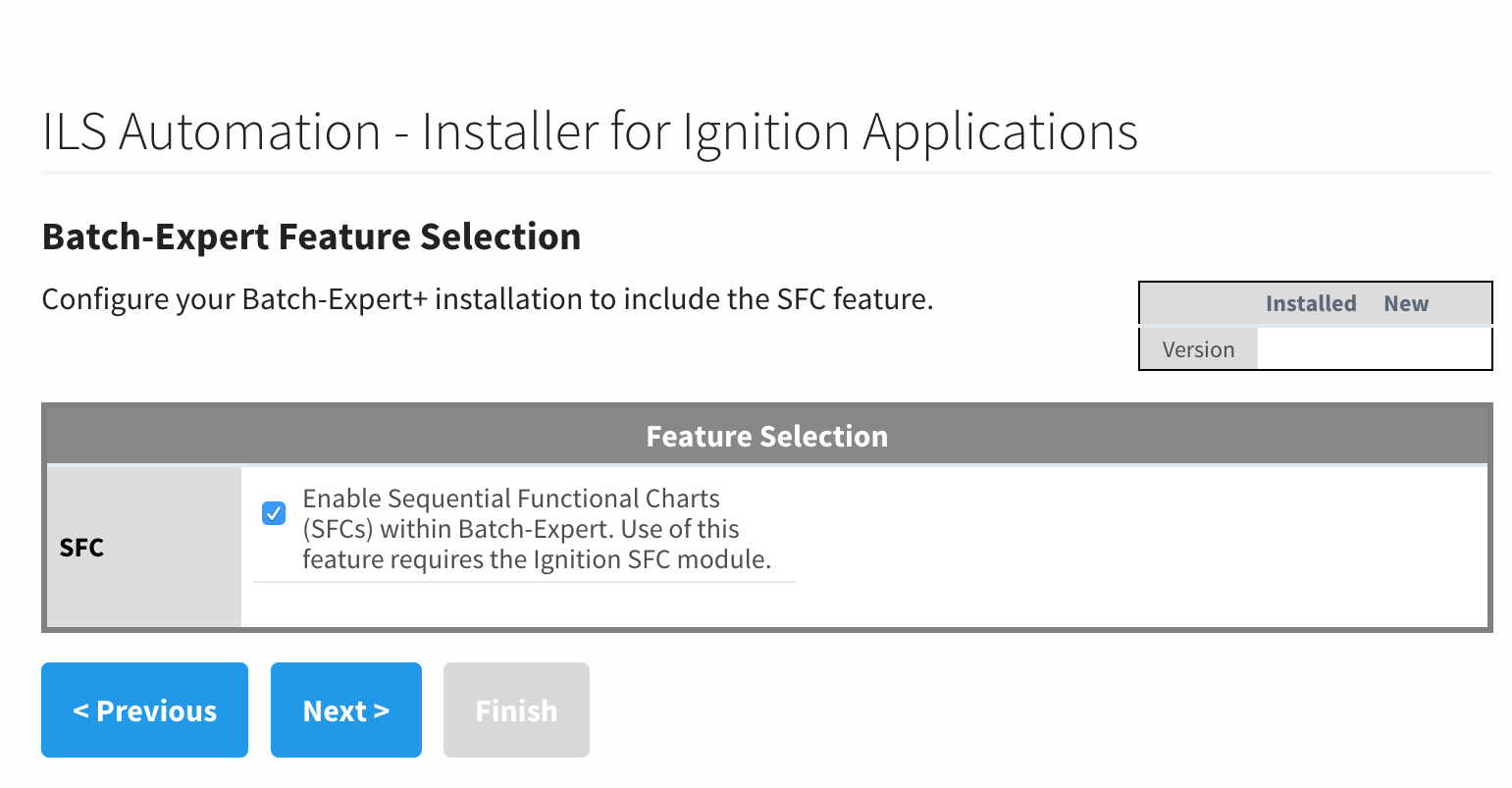


Figure –Feature Panel

## Interface Definition Panel

The purpose of this panel is to define the tag provider(s) and database connection(s) that will be used in subsequent project, database, transaction-group or scan-class panels. If any of these are referenced and the *site* panel is not used, then this panel is required.

The presence or absence of property elements determines the makeup of the resulting panel. “isolation” notation refers to tag providers and database instances that are used in a test, not production, environment.

<panel type=*"definition"* essential=*"true"*>

<title>Interface Definitions</title>

<preamble>Specify tag provider and database connections as

used by the Ignition Applications. This product requires that

both production and test mode versions are specified.

The selections hare are used by subsequent screens.

</preamble>

<!-- Allowed names are: database, DBMS, provider -->

<property name=*"database"* type=*"production"*></property>

<property name=*"database"* type=*"isolation"*></property>

<property name=*"provider"* type=*"production"*></property>

<property name=*"provider"* type=*"isolation"*></property>

</panel>

Allowable types for the properties are:

* production
* isolation
* secondary
* batchexpert
* pysfc

The form above uses and stores the selections as user preferences as well as storing into the internal Gateway database.

The property elements accept an optional script element. The value of this element is a path to a python module that accepts the value of the property as its single argument.

<property name=*"provider"* type=*"isolation">*

*<script>ils.common.properties.setProperty.setIsolationProvider*

*</script>*

</property>



Figure – Interface Definition Panel

It is important to note that the tag providers and database instances must be pre-defined before reaching this page. The pull-down selectors are populated with existing interfaces.

## Cleanup Panel

Previous installations may have created artifacts that are no longer used in the current. This panel defines a list of those artifacts and removes them.

<panel type=*"clear"* essential=*"false"*>

<title>Clear obsolete files</title>

<preamble>This step removes obsolete files installed during one

or more previous installations.

</preamble>

<!-- subtypes are: lib, user-lib, filesystem -->

<artifact name="Incorrect external jar" type="file"

subtype="lib">

<location>core/common/ils-common.gz</location>

</artifact>

<artifact name="Obsolete python module" type="directory"

subtype="user-lib">

<location>pylib/xom</location>

</artifact>

<artifact name="Unused test files" type="directory" subtype="filesystem">

<location>test/regression</location>

</artifact>

</panel>



Figure – Cleanup Panel

## Authentication Panel

The installer cannot create users and roles, but it can verify that the customer gateway includes an authentication profile that contains users with roles that are required for the application.

<panel type=*"authentication"* essential=*"false"*>

<title>Verify User Roles</title>

<preamble>Check authentication profiles to verify that at least

one of them includes users assigned to each role utilized

by the product.

</preamble>

<role name=*"operator"*>Operator - responsible for running

the application</role>

<role name=*"engineer"*>Application engineer - responsible

for configuring the application</role>

</panel>



Figure – Authentication Panel

This panel is advisory only.

## Icon Panel

If the application makes use of custom icons, then this panel may be used for its installation.

<panel type=*"icon"* essential=*"true"* version=*"1"*>

<title>Custom Icons for Ignition Applications</title>

<preamble>Custom icons are used extensively when drawing blocks

in the Block Language Toolkit.

They may also be found in many of the user screens.

</preamble>

<artifact name="Block Language Toolkit Icons" type="blt" subtype="icon">

<location>artifacts/icons/blt</location>

</artifact>

<artifact name=*"Application Icons"* type=*"app"* subtype=*"icon"*>

<location>artifacts/icons/custom</location>

</artifact>

</panel>



Figure – Icon Panel

## File Panel

The *file* panel facilitates copying files or directories from the release bundle to the file system. These directives require two sub-elements inside the artifact – *location* and *destination*.

The *location* element describes where to find the artifact within the bundle. The location may be either a single file or the root of a directory. Artifact type is either *text* or *binary*. When the type is *binary*, contents are copied exactly. Otherwise an attempt is made to modify the files for system-specific line terminations.

The *destination* element describes where in the file system the artifact is to be copied. Three destination options are defined (the artifact subtype):

* lib – the destination is relative to the Ignition installation “lib” area.
* user-lib – the destination is relative to the Ignition installation “user-lib” area.
* home - the destination is relative to the user’s home directory. In the case of Mac or Linux systems, the user is the installer of Ignition itself. On Windows systems, the “home” directory is C:\.

Note that there is no facility for copying files to an arbitrary location within the target file system. It is important to execute this panel before tags or projects since either of these may refer to external python.

Within the bill-of-materials .xml file, the \*nix convention of “/” for a file-separator is used when specifying artifact locations. This convention provides easier integration with *ant* and other tools.

<panel type=*"file"* essential=*"true"*>

<title>Install test files</title>

<preamble>This step adds test scripts to the filesystem inside the

user's home directory. It also initializes preferences

for the TestFramework control dialog.

</preamble>

<!-- Set preferences for the testing framework. If type is specified,

we manipulate the path. -->

<preference name="TestFramePreferences" key="TestDatasource"

type=*"home"*></preference>

<preference name="TestFramePreferences" key="TestScriptPath"

type=*"home"*></preference>

<preference name="TestFramePreferences" key="TestSetupPath"

type=*"home"*></preference>

<!-- paths are with respect to the file system root -->

<artifact name="Test python scripts" type="text" >

<location>artifacts/test/pylib</location>

<destination type=*"home"*>regression</destination>

</artifact>

<artifact name="Test framework scripts" type="text">

<location>artifacts/test/blt/basic</location>

<destination type=*"home"*>regression/blt</destination>

</artifact>

</panel>

Note the use of preferences that initialize file choosers in the installed application. These preferences relate to the user running the install.



Figure –File Panel

## Database Panel

There are two important scenarios when dealing with databases. In the first, there is no existing populated instance. The database is configured from scratch. An equivalent variation calls for existing tables to be dropped and re-created. All tables must then be populated with initial configuration data.

For databases, the artifact subtypes describe the script behaviors appropriate to these scenarios. Valid subtypes are:

* clear – the referenced script is expected to completely clear all the database of its contents. These include tables, views, triggers and stored procedures.
* create – the script creates and initializes all tables and views needed for the application. It does not create the instance.
* alter – the script alters the referenced instance in-place.

The example shown below contains a script to clear the database and a second script to populate. In the case of a new database instance, only the second should be run.

The DBMS may be specified as a property. This handles cases where there are constructs in the scripts that are DBMS-specific (and the installer knows about them). Valid options for DBMS are: ANSI, SQLSERVER, ORACLE, MYSQL and POSTGRES. Note the XML escaping to effect BOLD.

<panel type=*"database"* essential=*"true"* version=*"1"*>

<title>Database Create and Alter Scripts for the ACE Application

</title>

<preamble>This page provides the opportunity for either creating

a new database instance or altering an existing

instance to accommodate schema corrections. The

actions here are performed against the

&lt;b&gt;PRODUCTION&lt;/b&gt; instance.

</preamble>

<property name=*"database"* type=*"production"*></property>

<property name=*"dbms"* type=*"ORACLE"*></property>

<!-- Recognized subtypes are: clear, create, alter -->

<artifact name=*"ACE Database"* type=*"sql"* subtype=*"clear"*>

<location>artifacts/database/clearDatabase.sql</location>

</artifact>

<artifact name=*"ACE Database"* type=*"sql"* subtype=*"create"*>

<location>artifacts/database/createACE.sql</location>

</artifact>

</panel>



Figure –Database Panel

A second scenario, not shown, features a database alter script that is designed to make table structure adjustments without destroying existing table contents.

It is perfectly reasonable to expect a second database instance that is used for testing purposes. The XML is very similar to that found above. The panel is the same, with the exception that it targets a different data-source.

## Module Panel

If the installed product requires custom Ignition modules, use this panel to install them. This panel should be executed before tag or project definitions in either of these reference module script functions.

The artifacts are signed Ignition modules. Multiple modules may be specified.

<panel type=*"module"* subtype=*"production"* essential=*"true"* version=*"1"* >

<title>Install Custom Modules</title>

<preamble>This step installs modules necessary for the ACE control

application.

</preamble>

<artifact name="SFC Enhancements" type="module">

<location>artifacts/modules/ILS-SFC-module-unsigned.modl</location>

</artifact>

<artifact name="Application Toolkit" type="module">

<location>artifacts/modules/BLT-module-unsigned.modl</location>

</artifact>

</panel>



Figure –Module Panel

## Python Panel

The Python panel allows the installer to execute python scripts from the *user-lib/pylib* area of the Ignition installation. These scripts must exist before this panel can be executed, but it is possible to install them in a previous panel.

Up to three artifacts may be specified and a panel is created for each. The following artifact properties define the sub-panels:

* name – the name of the artifact is displayed as the sub-panel title
* type – the button label (title case)
* comment – the contents of this nested element is used as the text below the "Execute" button. Comment is a nested element.
* script – a nested element with a complete python path/module name. There are no arguments.

The example shown below contains a script to create or alter a database.

Note the XML escaping to effect BOLD in the preamble.

<panel type=*"python"* essential=*"true"* version=*"1"*>

<title>Database Create and Alter Scripts for ACE Application

</title>

<preamble>This page provides the opportunity for either creating

a new database instance or altering an existing

instance to accommodate schema corrections. The

actions here are performed against the

&lt;b&gt;PRODUCTION&lt;/b&gt; instance.

</preamble>

<!-- Recognized subtypes are: clear, create, alter -->

<artifact name=*"Create New Database Instance"* type=*"Clear"*>

<comment>Drop all tables in the database instance</comment >

<script > ils.db.clearDatabase</script >

</artifact>

<artifact name=*" Create New Database Instance"* type=*"Create"*>

<comment>Create tables that support the application</comment >

<script>ils.db.createDatabase</script >

</artifact>

</panel>



Figure –Python Panel

## ScanClass Panel

If the application makes use of a custom scan-class, then this panel may be used for its installation.

A scan-class is associated with a tag provider. The provider is specified in the <property> element of the bill-of-materials. If the *type* attribute is specified, then the provider that is selected on the interface definition panel used. If the attribute is missing, then the name found in the element value is used. This last option is discouraged as it amounts to “hard-coding” the provider name.

<panel type=*"scanclass"* essential=*"True"* version=*"1"*>

<title>Create alternative Scan Classes</title>

<preamble>Create scanclasses referenced by

&lt;b&gt;PRODUCTION&lt;/b&gt; tags.

Scanclasses determine tag poll rates.

</preamble>

<property name=*"provider"* type=*"production"*></property>

<artifact name="Production ScanClasses" type="scanclass" subtype="csv">

<location>artifacts/scanclasses/scanclasses.csv</location>

</artifact>

</panel>

The referenced artifact is a *.csv* file obtained from a scan-class export in the Ignition Designer.



Figure –ScanClass Panel

## Tag Panel

If the application requires its own tags, then this panel may be used to define them.

The tag artifacts are .xml files generated by exports in from the Ignition Designer. There may be multiple tag artifacts.

<panel type=*"tag"* subtype=*"production"* essential=*"true"* version=*"1"*>

<title>Install Tags and UDT Definitions</title>

<preamble>This page defines UDTs and tags that are needed

by the applications

when running in &lt;b&gt;PRODUCTION&lt;/b&gt; mode.

</preamble>

<property name=*"provider"* type=*"production"*></property>

<artifact name=*"UDT Definitions"* type=*"tag"* subtype=*"xml"*>

<location>artifacts/tags/UDTs.xml</location>

</artifact>

<artifact name="Tags Representing Connections" type="tag"

subtype=*"xml"*>

<location>artifacts/tags/connections.xml</location>

</artifact>

<artifact name=*"Production Tags"* type=*"tag"* subtype=*"xml"*>

<location>artifacts/tags/sqlTags.xml</location>

</artifact>

<artifact name=*"Vistalon Site Tags"* type=*"tag"* subtype=*"xml"*>

<location>artifacts/tags/siteTags.xml</location>

</artifact>

</panel>

Note the use of HTML escape sequences to create bold fonts.



Figure – Tag Panel

A common additional panel provides separate tags that are appropriate to the test environment. This scenario is not shown.

## Toolkit Panel

This panel is custom to ILS Automation. It installs values in the Gateway internal database into the *ILS\_TOOLKIT\_PROPERTIES* table. These values are used in various ways by ILS Automation products. This is, incidentally, the table used to store selected tag provider and database connection values.

<panel type=*"toolkit"* essential=*"true"* version=*"1"*>

<title>Update Toolkit Properties</title>

<preamble>Set properties in the internal database of the

Gateway. These particular parameters define extension functions

that update the database in response to operator operations on

diagrams.

</preamble>

<property name="com.ils.application(property-get-script)">

ils.extensions.appProperties.getAux</property>

. . .

</panel>



Figure –Toolkit Panel

## Project Panel

The project panel should be near the end of the installation sequence. The reason for this is that projects may be dependent on other artifacts (such as external python or modules) to start cleanly.

For projects, the artifact subtype attributes control the creation and update behavior for the selected project. Valid subtypes are:

* full – the internal artifact is installed as a new project, completely replacing any existing project with the same name.
* global – the internal artifact is a partial project which will be merged with the current global project. The internal project’s resources will overwrite any resources that exist both projects
* partial – the internal artifact is a partial project which will be merged with the selected existing project. The internal project’s resources will overwrite any resources that exist both projects.

The figures below show several panel configurations. The .xml fragment corresponds to the second case – where both full and partial project options are shown. Multiple artifact elements result in multiple sections in the resultant panel.

<panel type=*"project"* subtype=*"main"* essential=*"true"* version=*"1"*>

<title>

Install a New ACE Project or Merge Changes into an Existing Project

</title>

<preamble>ACE is the main Ignition project. This page provides

artifacts for a clean, new installation or, alternatively,

a merge with an existing version. A merge preserves any user

modifications.

</preamble>

<!-- For projects, artifact subtypes are: full, partial and global -->

<artifact name=*"ACE"* type=*"project"* subtype=*"full"*>

<location>artifacts/projects/ACE.proj</location>

</artifact>

<artifact name="ACE Partial Project" type="project" subtype="partial">

<location>artifacts/projects/ACE.proj</location>

</artifact>

</panel>



Figure – Project Panel



Figure – Project Panel (Global)

If the backup checkbox is selected, then any existing project that is the target of a merge or replacement will be renamed with a unique suffix and disabled.

## Documentation Panel

Documentation is a non-essential panel. The screen allows the user to select individual documents to download. The display compares the current version with that of the last-downloaded version of the document. This makes it clear which of the available documents have changes in this release.

Note that each document artifact includes a release attribute. Each artifact is downloaded separately.

<panel type=*"documentation"* essential=*"false"*>

<title>Product Documentation</title>

<preamble>The list below contains both user-level and design documents.

They may be downloaded individually.

</preamble>

<artifact name="Installation Manual" type="documentation"

subtype=*"docx"* release=*"0.12"*>

<location>artifacts/doc/InstallationManual.docx</location>

</artifact>

<artifact name="Application Testing Framework" type="documentation"

subtype=*"docx"* release=*"1.0"*>

<location>artifacts/doc/ApplicationTestFramework.docx</location>

</artifact>

. . .

</panel>



Figure –Documentation Panel

## Source Panel

Like the previous, this is an optional panel. It is used for products in which source code is part of the distribution. As with documentation, the user is allowed to select which, if any, of the source bundles is to be downloaded. The browser determines the download location.

<panel type=*"source"* essential=*"false"* version=*"1"*>

<title>Source Distribution</title>

<preamble>The Java source for the Application Toolkit and extensions

to Sequential function charts are bundled into a single zip file.

The source distribution comes complete with Eclipse project files

and ant scripts for building. Python source is available directly

from its installation in the /user-lib/pylib sub-directory of the

Ignition installation.

</preamble>

<artifact name="Toolkit Source Code" type="source" subtype="zip"

release=*"0.1r15"*>

<location>artifacts/source/blt-0.1r15.zip</location>

</artifact>

<artifact name="SFC Extensions Source Code" type="source" subtype="zip"

release=*"0.1r15"*>

<location>artifacts/source/sfc-0.1r15.zip</location>

</artifact>

</panel>



Figure –Source Code Panel

We’ve called this a panel for source code, but it can be used to convey any informational file of interest to the user.

## Summary Panel

The summary panel marks the end of the installation. It lists a summary of the installation state of all required artifacts.

<!-- The summary panel must always be displayed, thus no version -->

<panel type=*"summary"* essential=*"true"*>

<title>Installation Summary</title>

<preamble>The list below shows which of the installation steps

have been completed. Selecting "Finish" will record the current

version of the ACE Ignition Applications in the Gateway internal

database.

</preamble>

</panel>



Figure –Summary Panel

## Epilog

Once the user presses “Finish”, a final panel is shown. It has no navigation links. While it is being shown, the installation module is deleted and the installation link is removed from the Gateway’s configuration panel.

<!-- The completion page is displayed on finish.

Use it to display post-install instructions. -->

<epilog>

<title>Installation Complete</title>

<preamble>

The automated installation and configuration

of the ACE Control Ignition Application is complete.

There are just a few more manual steps required.

</preamble>

<note name=*"Disabled Projects"*>Newly created projects are

installed disabled. They must be enabled before use.</note>

<note name="Gateway Restart">

Several modules with active components have been installed.

A Gateway restart is required.</note>

</epilog>

Figure –Epilog