Application Platform

Users Guide

Version 9.8

April 2015

This document applies to webMethods Application Platform Version 9.8 and to all subsequent releases. Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

DOCUMENT ID: PLD-UG-98-20150415

CONTENTS

1 About this Guide

Software AG Application Platform IDE components are installed as a set of features within Software AG Designer. Online help is included in Designer Guide node of the Eclipse Help table of contents. Expand this node to view the available help sets. If a feature is not installed, there will be no help set available for it. However, you can view PDF versions of all designer features on the Software AG Documentation website.

1.1 Document Conventions

Convention	Description
Bold	Identifies elements on a screen
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in text.
{ }	Indicates a set of choices from which you must choose one. Type only the information found inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of the choices only. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
…	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (…).

1.2 Online Information

This document and other Software AG documents mentioned in this guide may be found at the Software AG Documentation website.

2 Introduction

The Application Platform consists of a collection of components including development tools and runtime components. Java applications are constructed using Designer as the main development tool. The applications are deployed as OSGi bundles into a dedicated development server. A design goal for the product is to provide the means for developers to create projects with as little OSGi knowledge as possible. Nonetheless, developers are strongly encouraged to become familiar with OSGi given it is a foundational cornerstone if the Software AG server architecture.

This guide is organized into following areas of interest:

- Introduce the architecture
- Development environment setup pre-requisites
- Visually describe the development activities with emphasis on Designer's Application Platform tooling
- Briefly discuss activities required to deploy and configure projects outside of development setting
- Finish up with a miscellaneous assortment of troubleshooting topics

2.1 Architecture

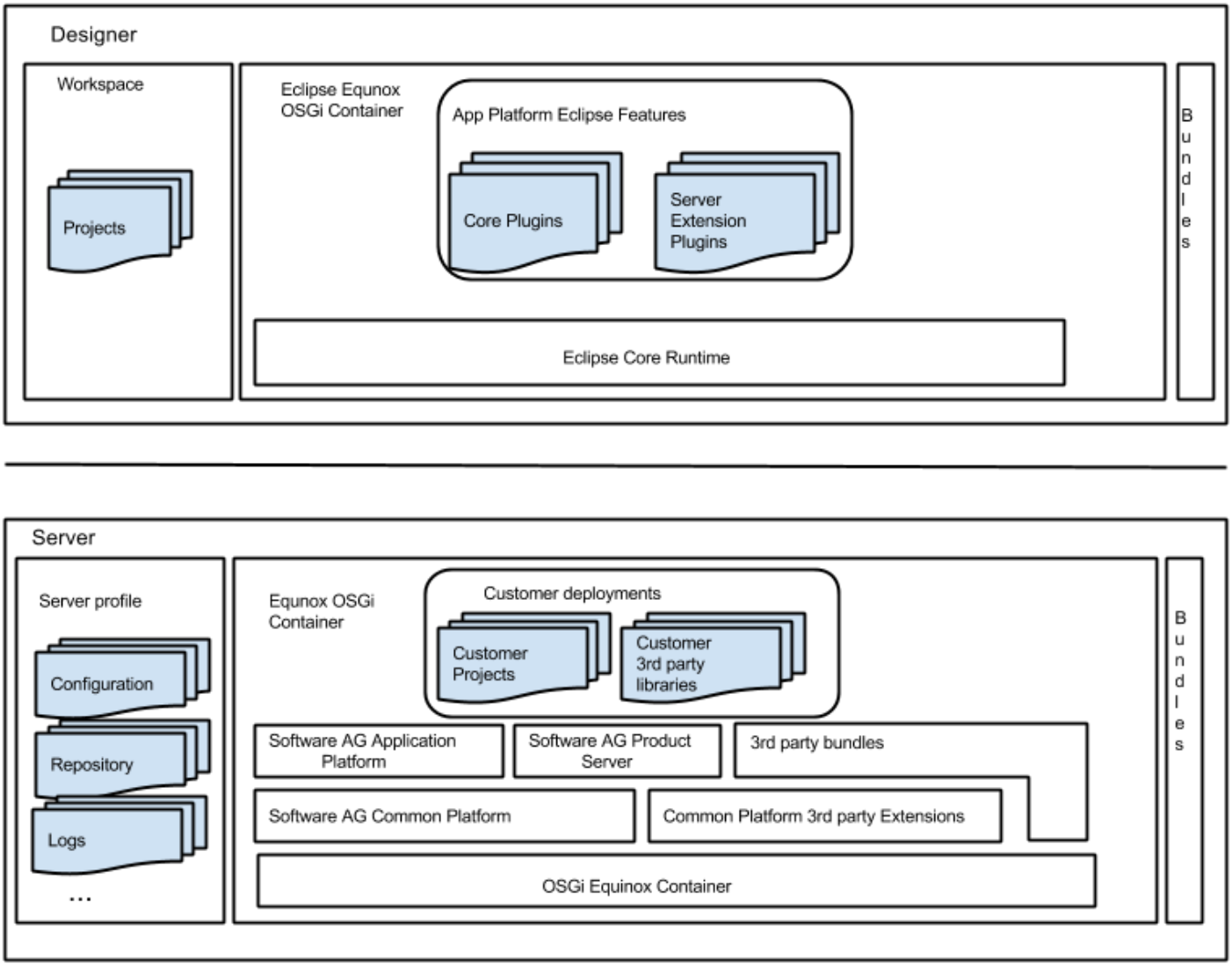
The architecture is based upon a composite list of integrated components as illustrated below. The sections that follow briefly describe each component and often provide references to more documentation for each component.

2.1.1 Diagrams

The architecture can be visualized in several different contexts.

2.1.1.1 Component Stack

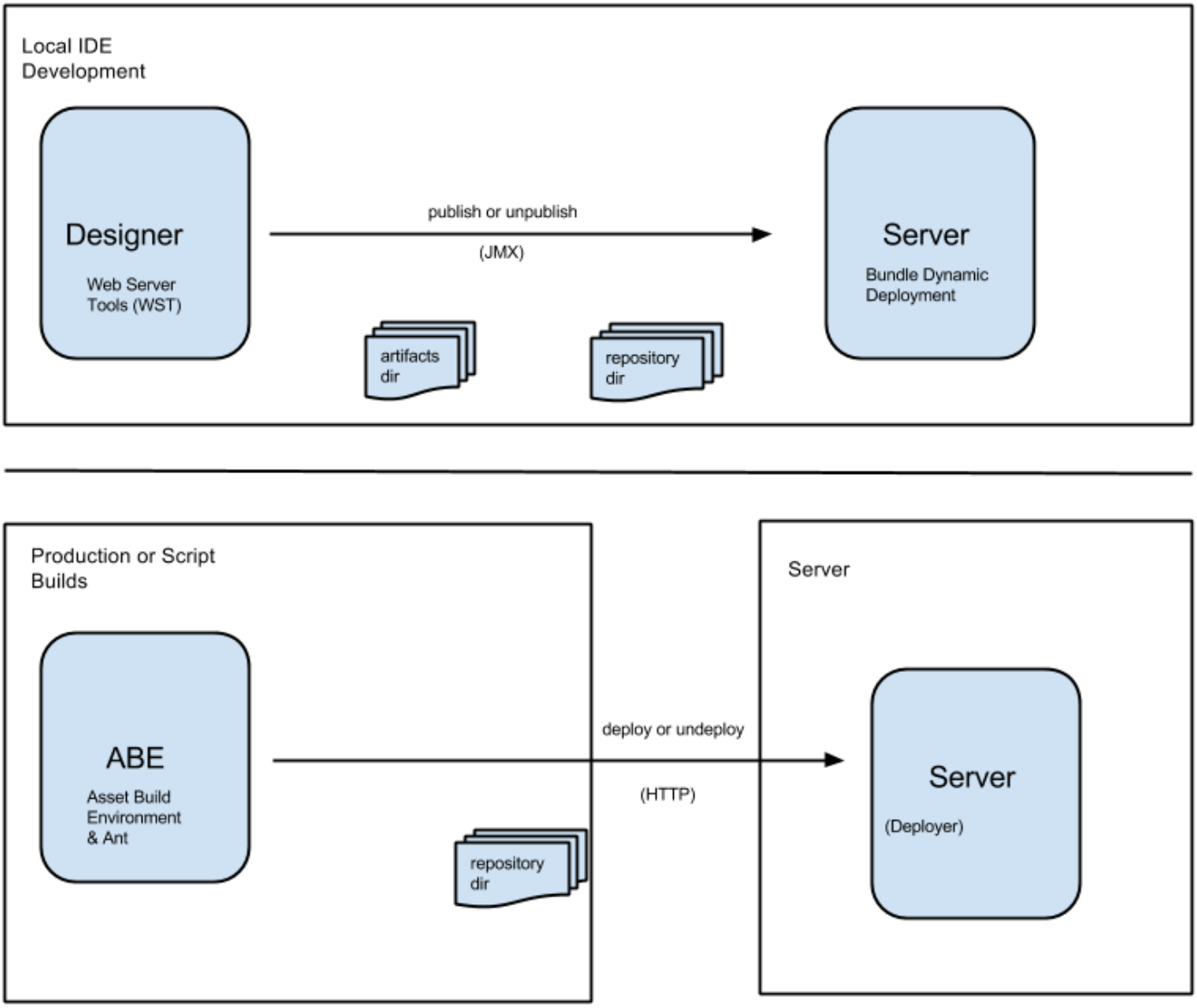
At a lower level, the development IDE and server represent containers which manage encapsulated components.



2.1.1.2 Bundle Deployment

This publish model applies to both project source as well as 3rd party bundles.

Each large rectangle represents a different physical computer. Each shaded rectangular shape represents a Java VM.

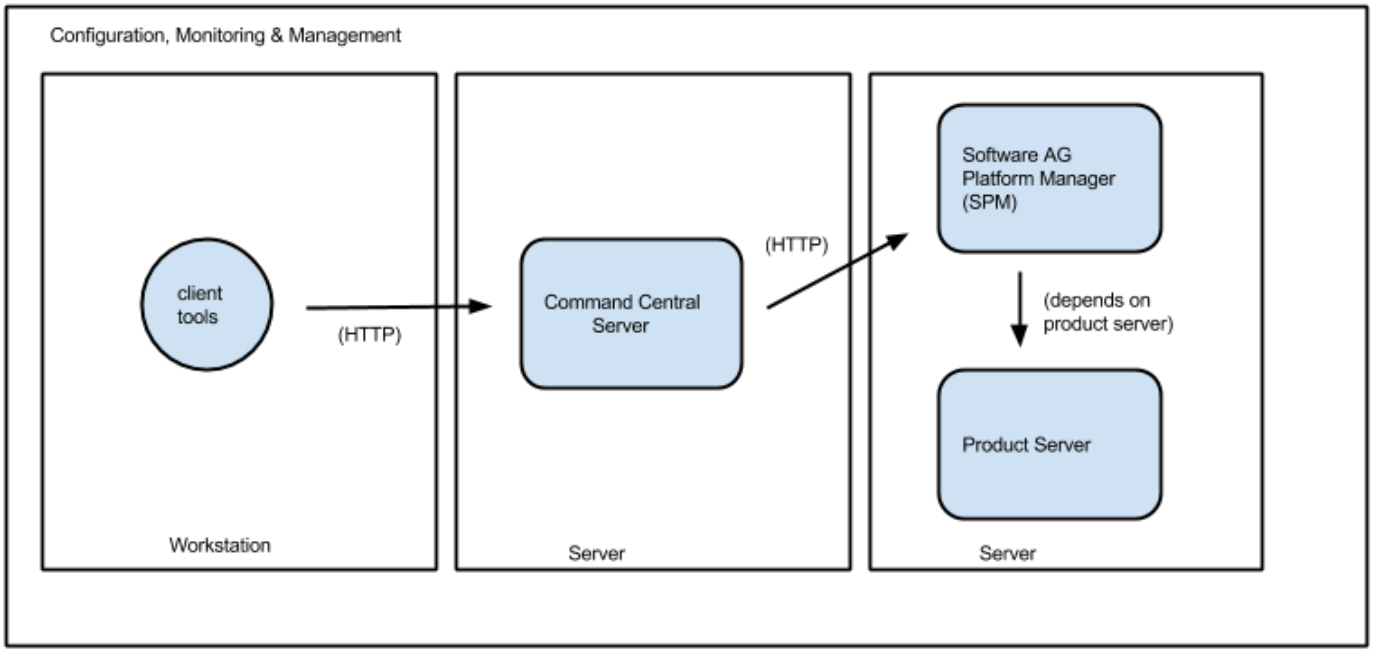


2.1.1.3 Application Platform Configuration

In the diagram below, we have the following components:

- Client tools (Web browser or Command Line)
- Command Central Server
    - The server may be configured to manage a group of SPM nodes in the customer landscape
- SPM running in its own dedicated JVM
- Product server hosting the customer projects

Each large rectangle represents a different physical computer. Each shaded shape represent a Java VM.



2.1.2 OSGi Foundation

OSGi plays a crucial role in the Application Platform's architecture. In OSGi, a deployment module is typically a Java jar file containing a META-INF/MANIFEST.MF file with additional headers. This special jar file is called a bundle by convention. The metadata in these headers is used by an OSGi container when it is time to install the bundle into a server. Equinox (also used by Eclipse) is the OSGi container implementation found in Application Platform-supported servers. It is important to understand the distinction between a plain Java jar file versus an OSGi bundle. The Application Platform tooling creates and installs bundles into the server. Generally, 3rd party dependencies must be bundles or use Application Platform tools to create bundles from a simple jar; however, it is possible to embed a simple jars if included in the "lib/" folder. Please see the Lib Folder section for more details.

2.1.3 Designer IDE

Designer is the supported IDE for building Application Platform components. Additional Eclipse features each with its own set of plugins are included in the product. Some examples of the features supported are as follows:

- Project wizards for creating Java and Web applications
- Eclipse server tools (WST) integration for publishing and debugging projects in the server
- Dialog wizards to create Java bindings to server components

- Custom perspective and views
- Various utilities to assist with project development in the OSGi environment

For more Designer general details outside of the Application Platform scope, please refer to the "*Working with Software AG Designer*", or one of the product-specific guides such as "*Service Development Help*"

2.1.4 Server

Software AG servers capable of receiving Application Platform projects have the following characteristics:

- based upon OSGi runtime containers
- contain a collection of Software AG product bundles called the Software AG Common Platform
- managed by a similar set of scripts and files to manage cross-cutting concerns in a consistent manner. Some examples include the following:
    - Logging,
    - Configuration,
    - Lifecycle scripts

For more details, please refer to the following guides:

- "*Working with the webMethods Product Suite and the Java Service Wrapper*"
- "*Working with Software AG Runtime*"
- Product-specific guides such as "*Integration Server Administrator's Guide*".

2.1.5 Deployer

After completing the development phase, projects may be deployed into servers using command line scripts via Software AG Deployer

- Projects are built and packaged as assets using repeatable processes via the Asset Build Environment (ABE)
- These assets are deployed to a one or more target systems using Deployer

For more details regarding Deployer or ABE, please refer to the "*Deployer User's Guide*" cross-product document.

2.1.6 Platform Manager and Command Central

Every server may participate in a common infrastructure for managing configuration and monitoring product statuses.

- Each product has a manager running in a separate process called the Software AG Platform Manager (SPM)
- A centralized product called Command Central to administer a collection of products across networked servers via client tools
    - Command Line Interface
    - Web Browser-based GUI

For more details regarding SPM, please refer to the "*Working with Software AG Runtime*" cross-product document.

3 Development Activities

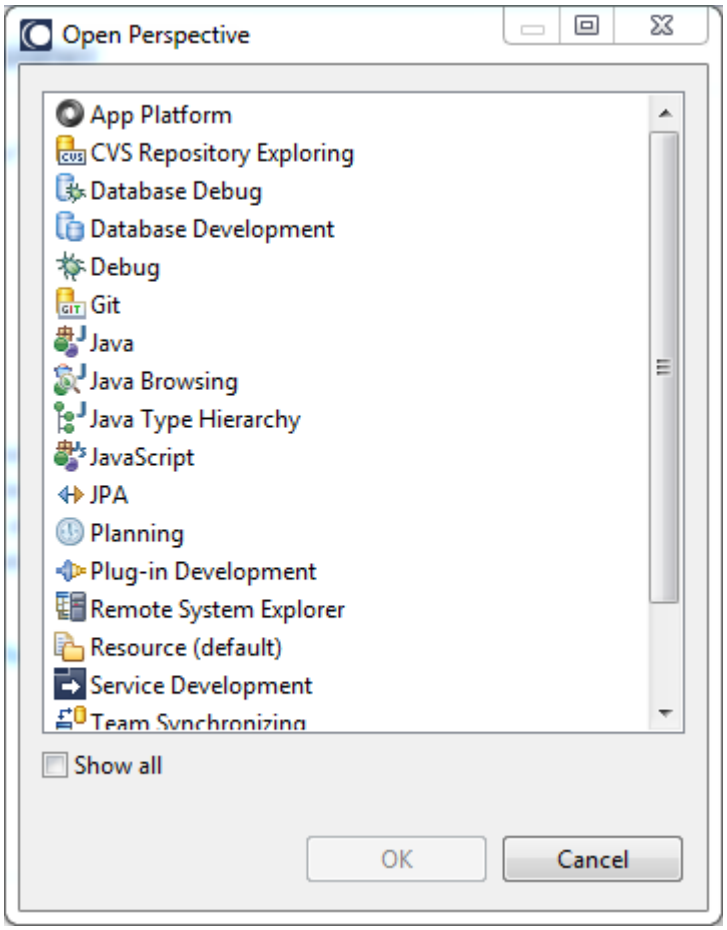
This section describes the features added to Designer to support Application Platform development.

3.1 Getting Started

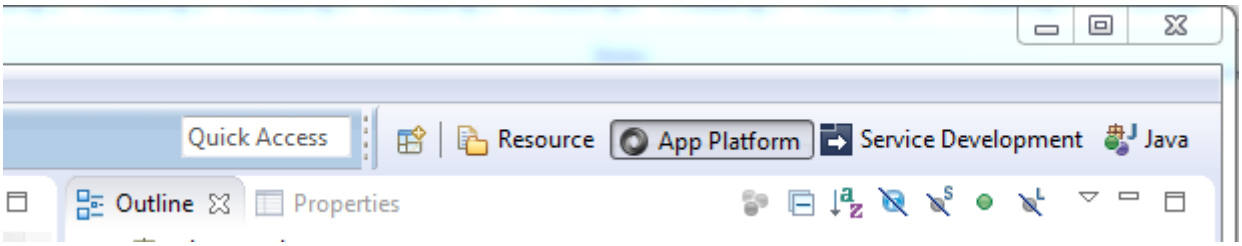
After installing Application Platform, the following sections discuss tasks that should be performed immediately before building projects.

3.1.1 Open Application Platform Perspective

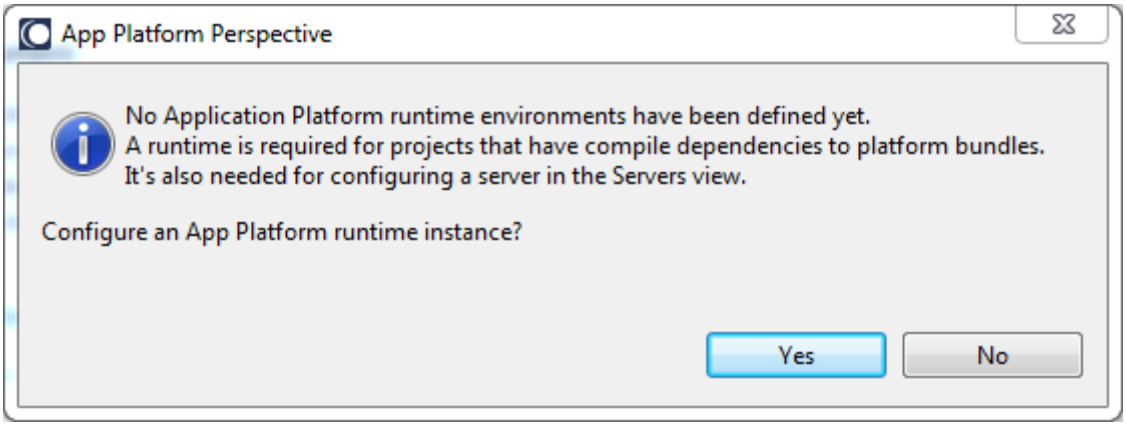
Application Platform has a dedicated Eclipse perspective. This perspective contains the basic views which are most often used when developing applications. In the Window menu, select the Open Perspective submenu and then click on the *Other...* menu item. Select the App Platform perspective from the list.



After the perspective is opened first time, it will be cached for quick access.



Each time the perspective opens, a warning dialog is presented to remind the user to create a runtime environment if one does not exist. A runtime environment must be configured before creating a project or publishing to the server.

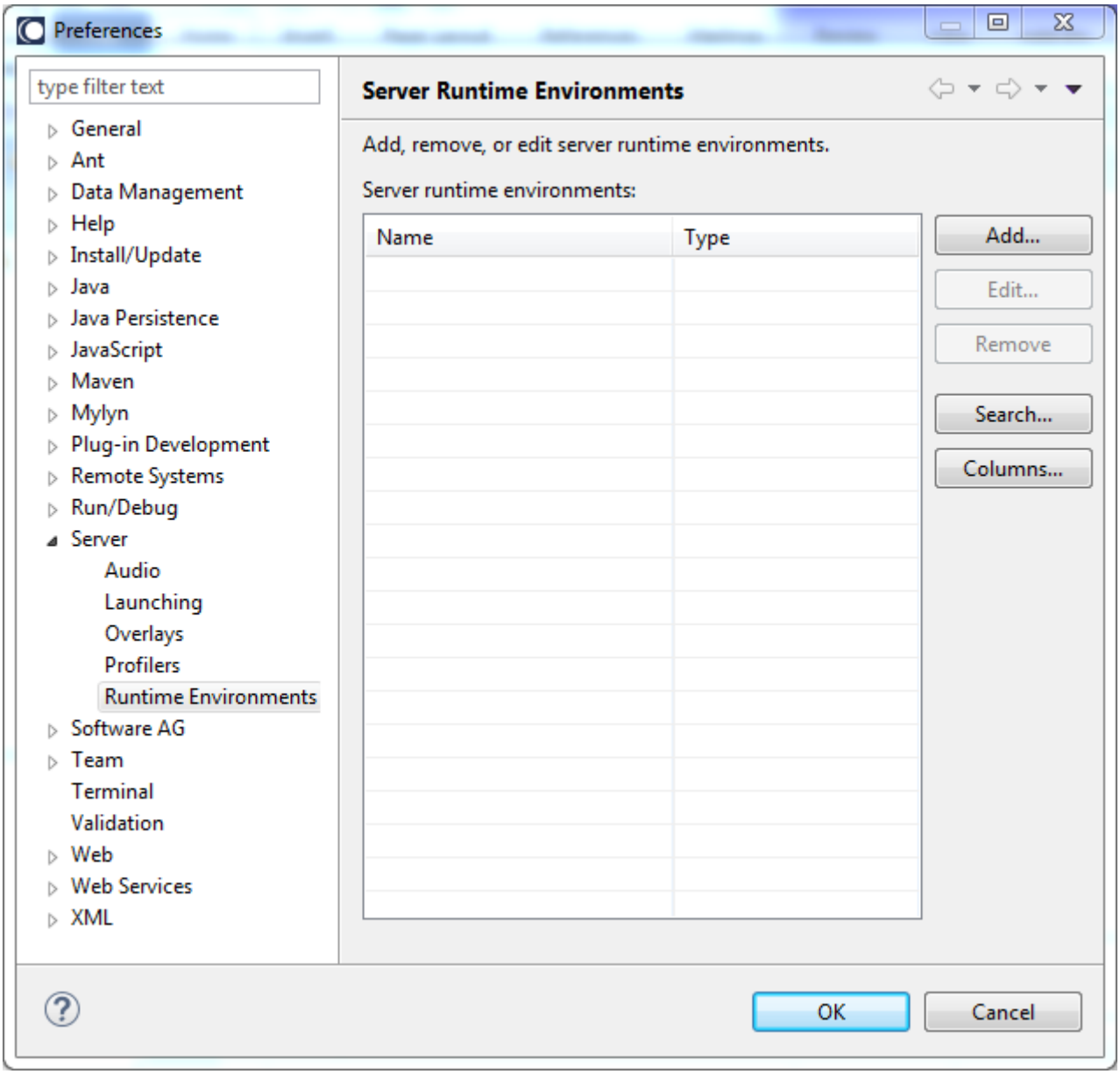


Pressing the Yes button will take the user to the App Platform Runtime configuration view as discussed in the next section.

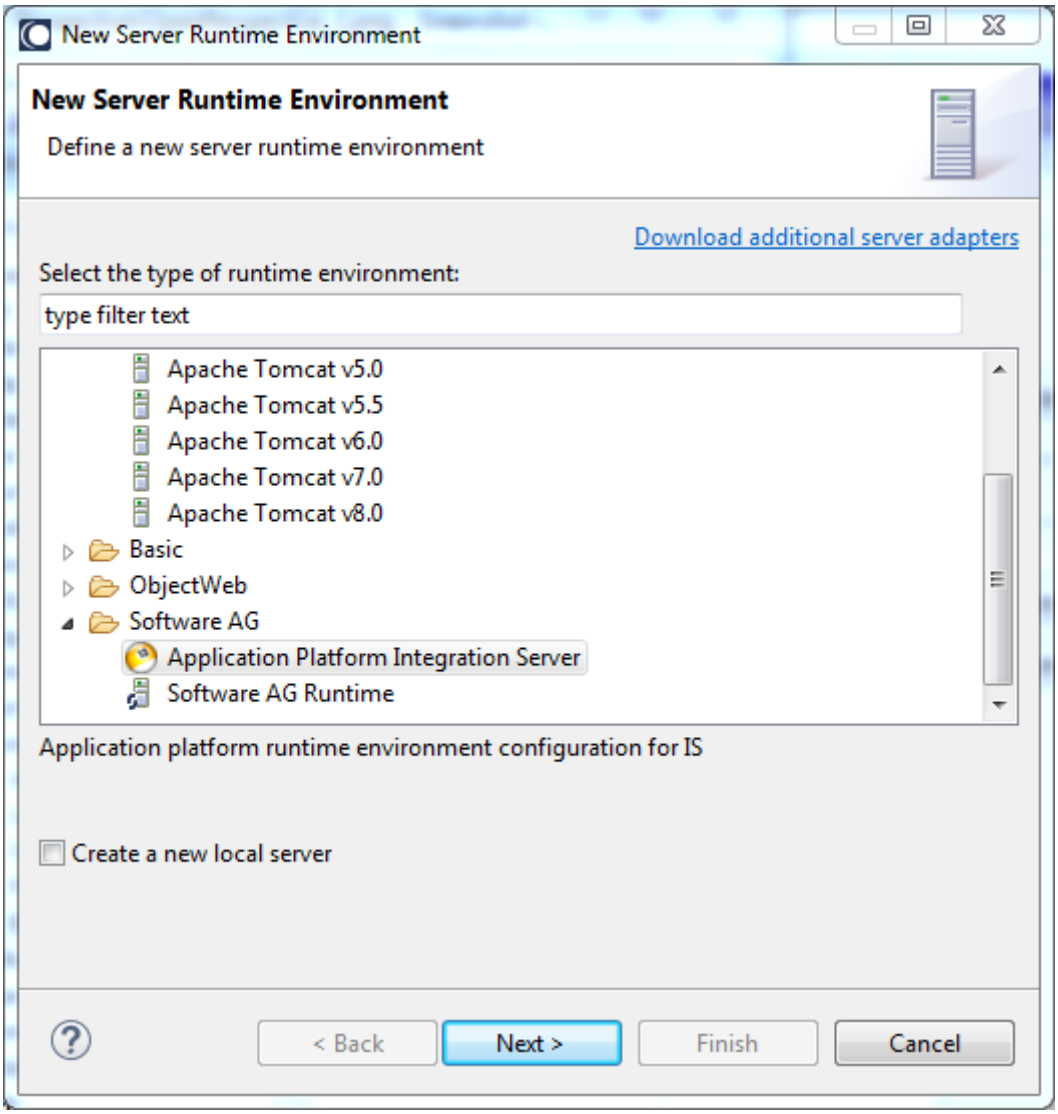
3.1.2 Server Runtime Environment

The Application Platform must have a server runtime configured so projects may reference its runtime container. Runtime containers are Eclipse configuration elements that define a set of product libraries a project should include in its classpath. An absolute path to the product installation is required for the runtime configuration. Users cannot change the bundles included in the runtime. Initially, the preference view will not have any runtimes defined, so the first step is to define one.

The configuration screen may be accessed from Designer by selecting the Window/Preferences/Server/Runtime Environments menu item. Navigate to this preference panel and click the Add button.

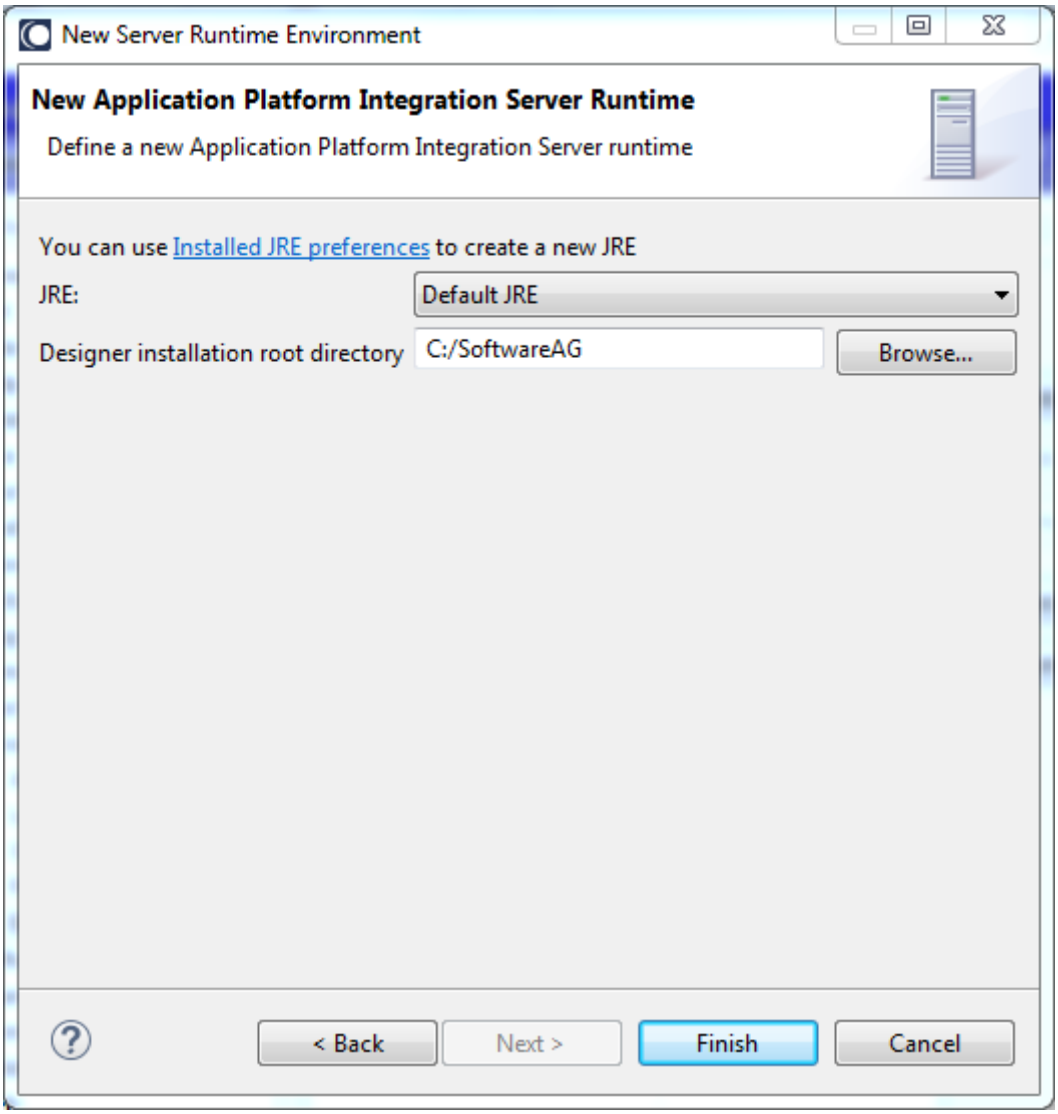


Scroll down and choose the Application Platform server for the server product. E.g. *Application Platform Integration Server.*



Note: The only server available for the 9.8 release is the Integration Server.

Provide the path to the root directory Software AG installation folder.



Note: This configuration pane will not enable its Finish button until a valid root directory has been selected – i.e. a root directory containing a *profiles* directory.

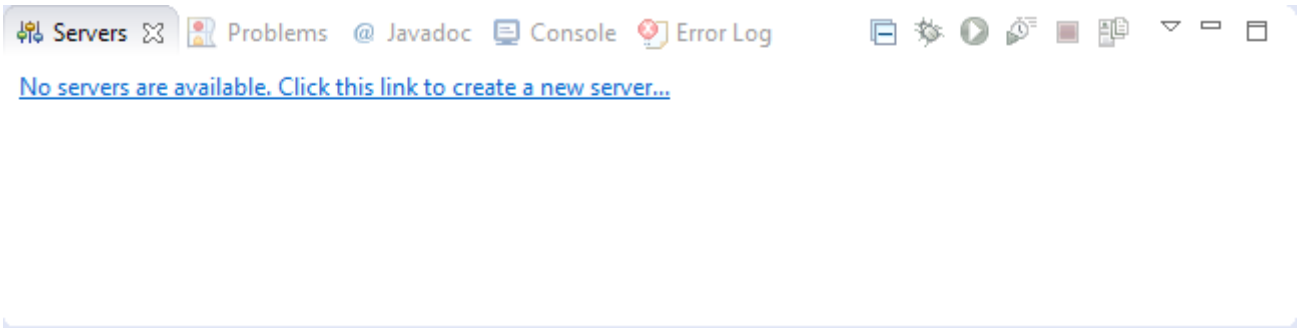
Important: The Designer installation root directory configuration is stored in the Eclipse workspace metadata area. If installing another instance of Designer on the same computer, you should not share the same workspace directory. Doing so can lead to errors since both installations will be sharing the same runtime configuration and communicating to the same server.

Click the Finish button to complete the configuration step.

3.1.3 Server Configuration

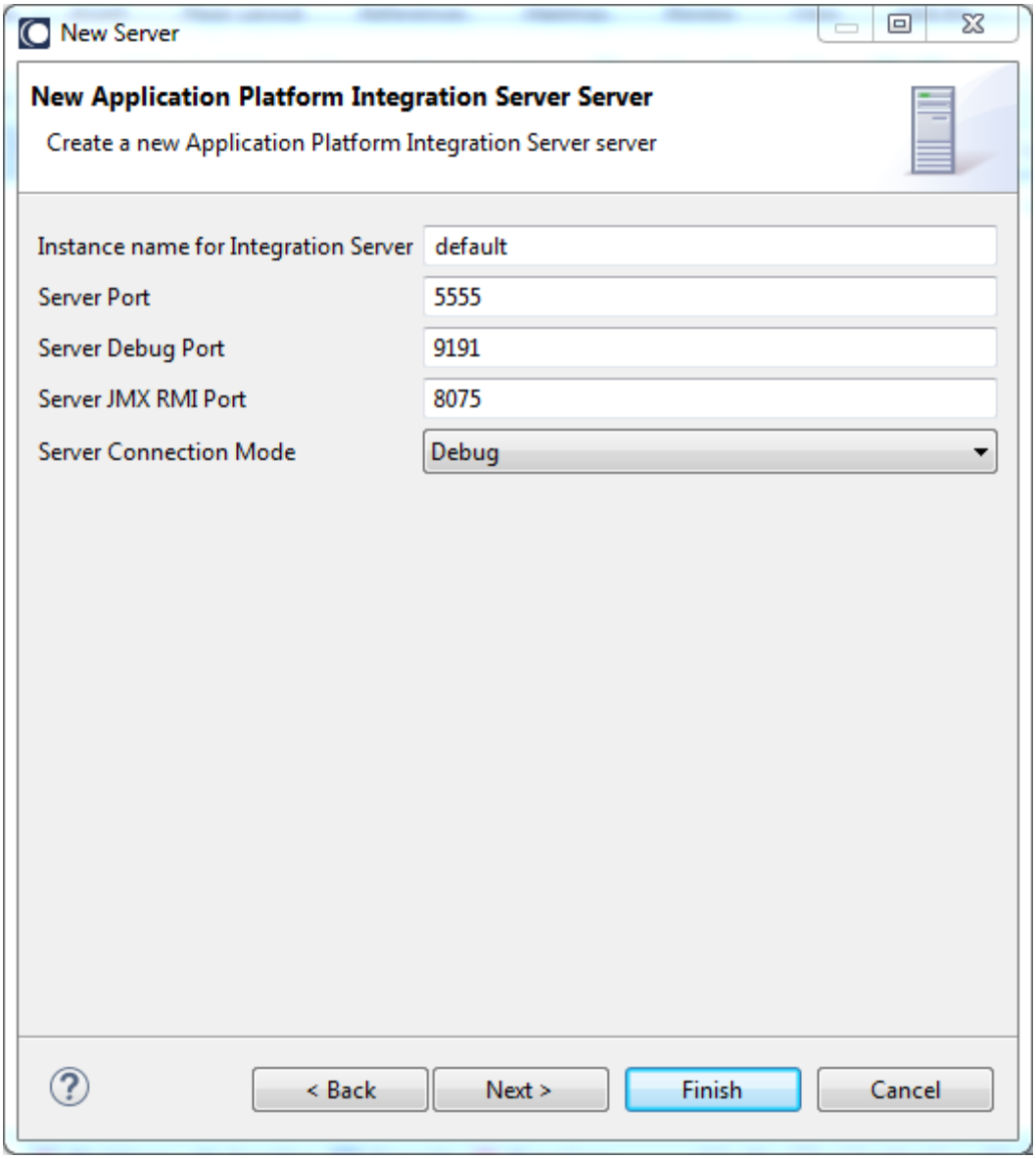
Once a Runtime Environment has been created and the dialog is closed, a server configuration must be created before projects may be published. Server configurations are managed from the *Servers*

view located at the bottom of the App Platform perspective. The first time the *Servers* view is opened, a link is available to create a new server.



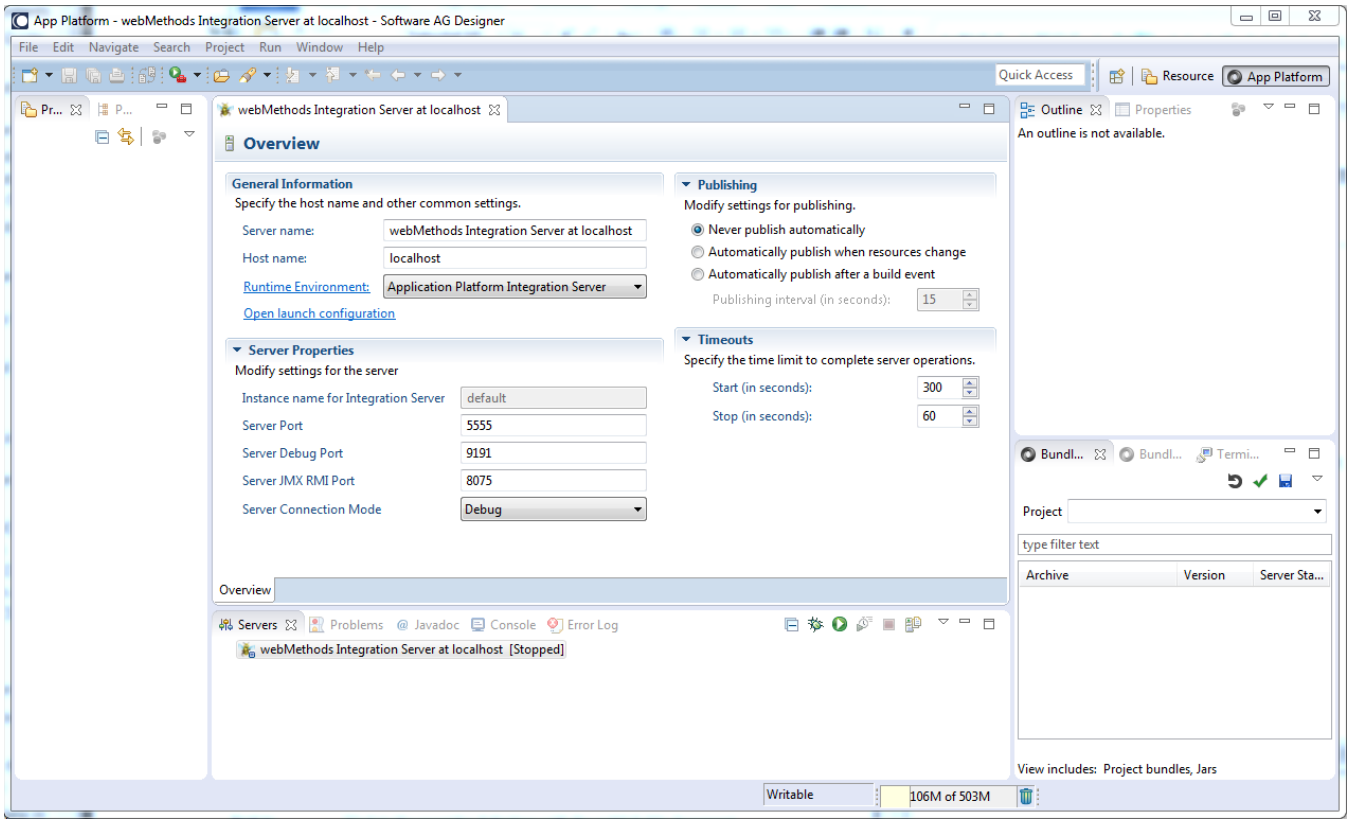
Click the link to create a server. The default values should be correct if none of the server properties were changed during installation.

Important: The server should be installed as a "application" rather than a "service" as described in the installer. In this situation, errors can occur due to a mismatch between the service and Designer's configuration used to launch the server. Please see the Server is installed as a service section for more details.



Important: The Instance name must be *default* for this release.

Here is a completed server configuration. Once a runtime and server configuration have been created, it is possible to manage the development server from Designer. For example, the server may be started and stopped by clicking the appropriate toolbar action icon or from a context menu by right-clicking the server in the view. For more details, please refer to the *Using the Server Tools* section of the *Web Tools Platform User Guide* in Designer's Help Contents.



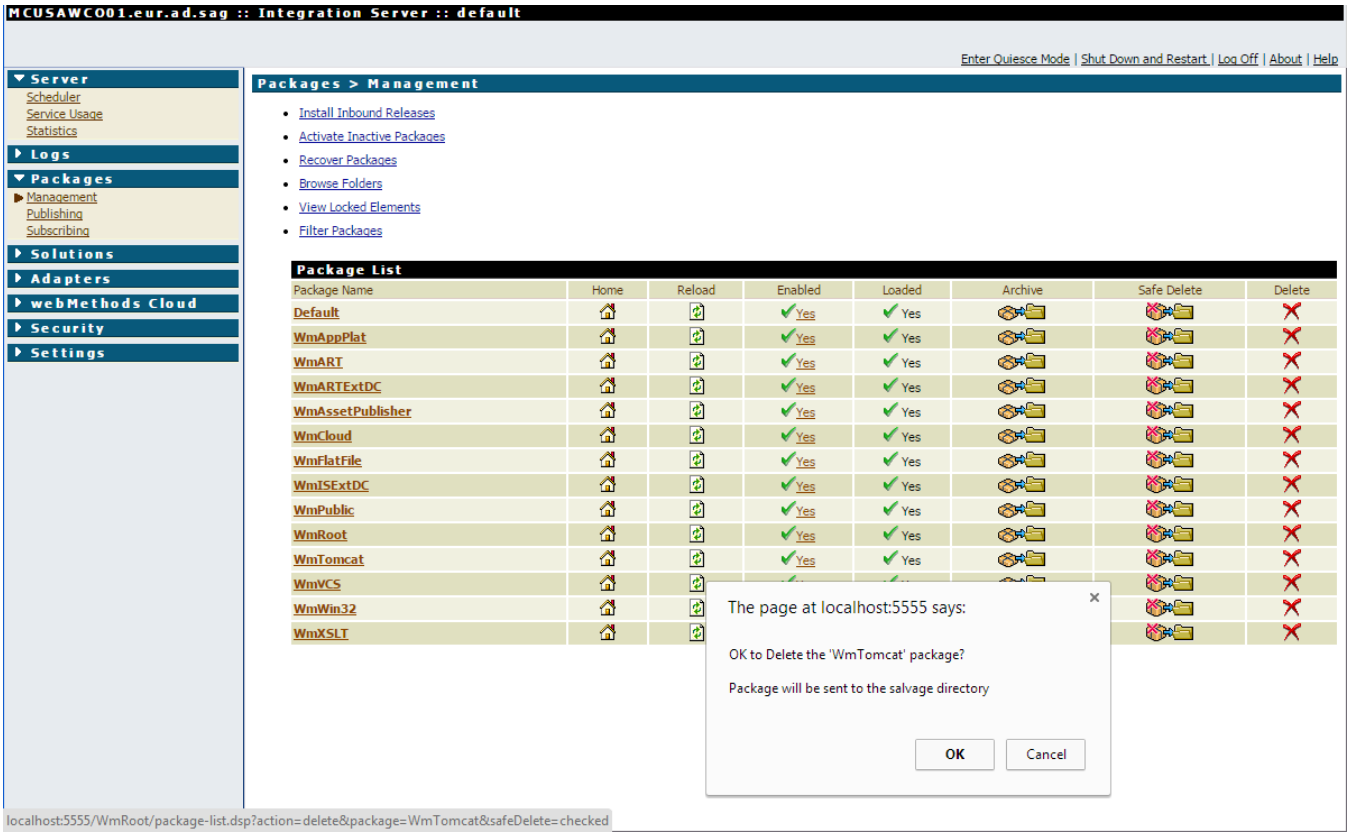
3.1.4 Integration Server Considerations

The following tasks are required to ensure a functional environment for publishing bundles into the Integration Server.

3.1.4.1 Disable the WmTomcat package

The Application Platform uses the Software AG Web Server included in the Software AG Common Platform environment. The Integration Server also provides its own Tomcat instance via the WmTomcat IS core package. This package must not be enabled in a server profile hosting Application Platform web modules.

Using the Integration Server Administrator utility, delete or disable the WmTomcat package, and restart the server. Consult the "*webMethods Integration Server Administrator's Guide*" for more details.



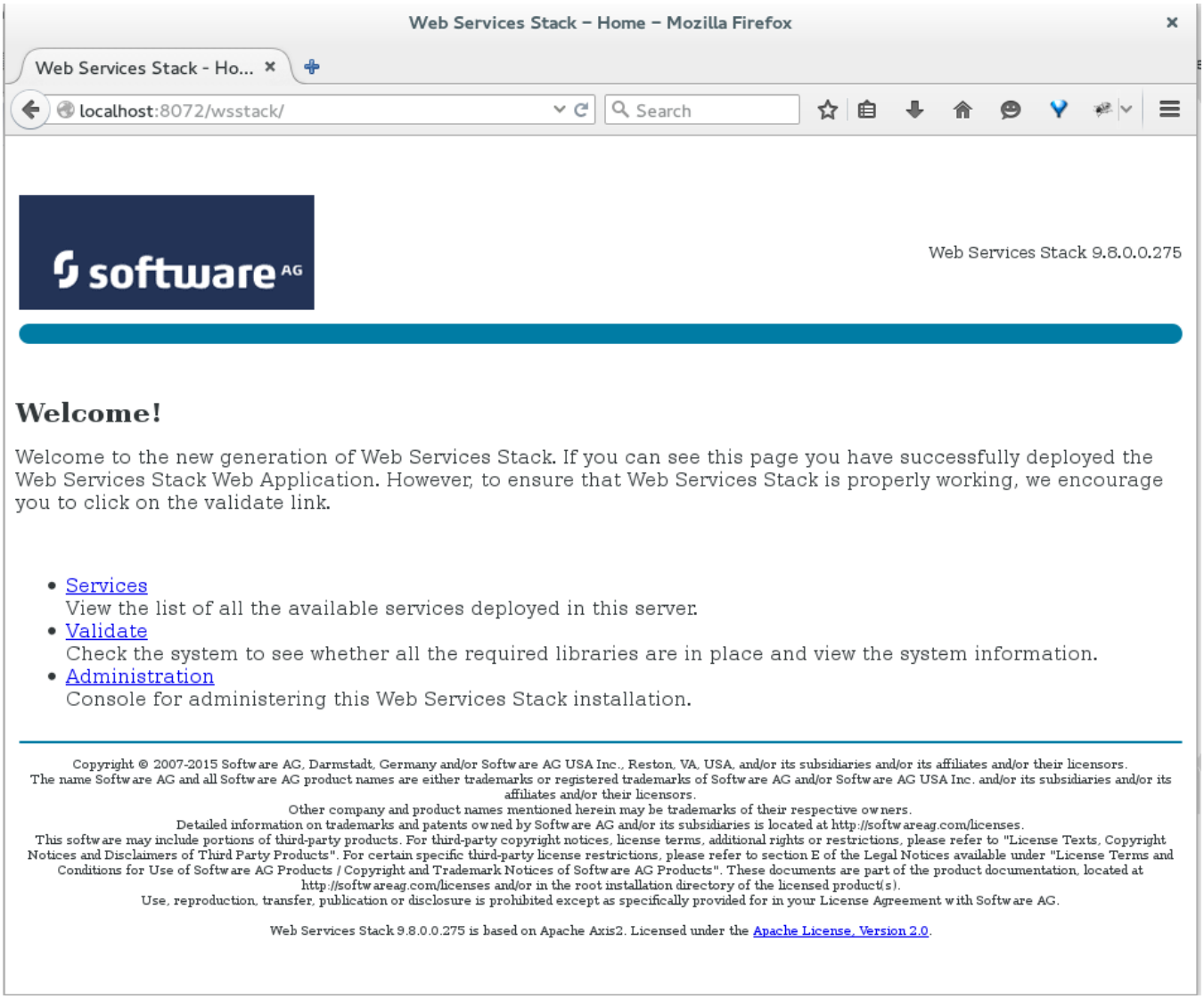
3.1.4.2 *Common Tomcat Platform*

Application Platform uses a Software AG common platform component called the Common Tomcat Platform (CTP). This component is enabled when Application Platform is installed.

Note: The Integration Server's profile is modified during the installation process by updating the ${sagHome}/profiles/IS_default/configuration/config.ini file. The config.ini file should contain this entry "com.softwareag.platform.catalina.launcher.gemini.skip=true" when the Application Platform is installed.

The default ports are 8072 for HTTP and 8074 for HTTPS. The Tomcat configuration file is located here : ${sagHome}/profiles/IS_default/configuration/tomcat/conf/server.xml.

To confirm CTP's availability after installation, start the Integration Server installation, and enter http://localhost:8072 in a web browser.



3.1.5 Optional Configuration

This section discusses additional tasks to perform; however, these steps are not required.

3.1.5.1 Eliminate NLS Warnings in Designer Error View

Eclipse may produce warning messages for any localized messages which go unused. This can lead to situations where many log messages are generated with the following format:

Warning: NLS unused message: {resource key} in: {file reference}

These messages do not indicate a problem with the installation. They may be suppressed by defining a system property for the Designer instance via its eclipse.ini file. Simply add the following property at the end of the eclipse.ini file.

-Dosgi.nls.warnings=ignore

Note: The eclipse.ini file is found in the *${sag.install.dir}/Designer/eclipse/* directory path.

3.1.5.2 Enable OSGi Console for Server Profile

Once developers are more familiar with OSGi, it can be helpful to use the OSGi console to assist in troubleshooting efforts, or simply as a means to gain additional insight into OSGi. Eclipse's Terminal view is included in the App Platform perspective; however, it requires additional configuration. Please refer to the OSGi Console section for more details.

Note: The OSGi console must be enabled if using the terminal view in the Application Platform perspective.

Important: The OSGi console uses unsecured telnet. This console should be disabled for production systems.

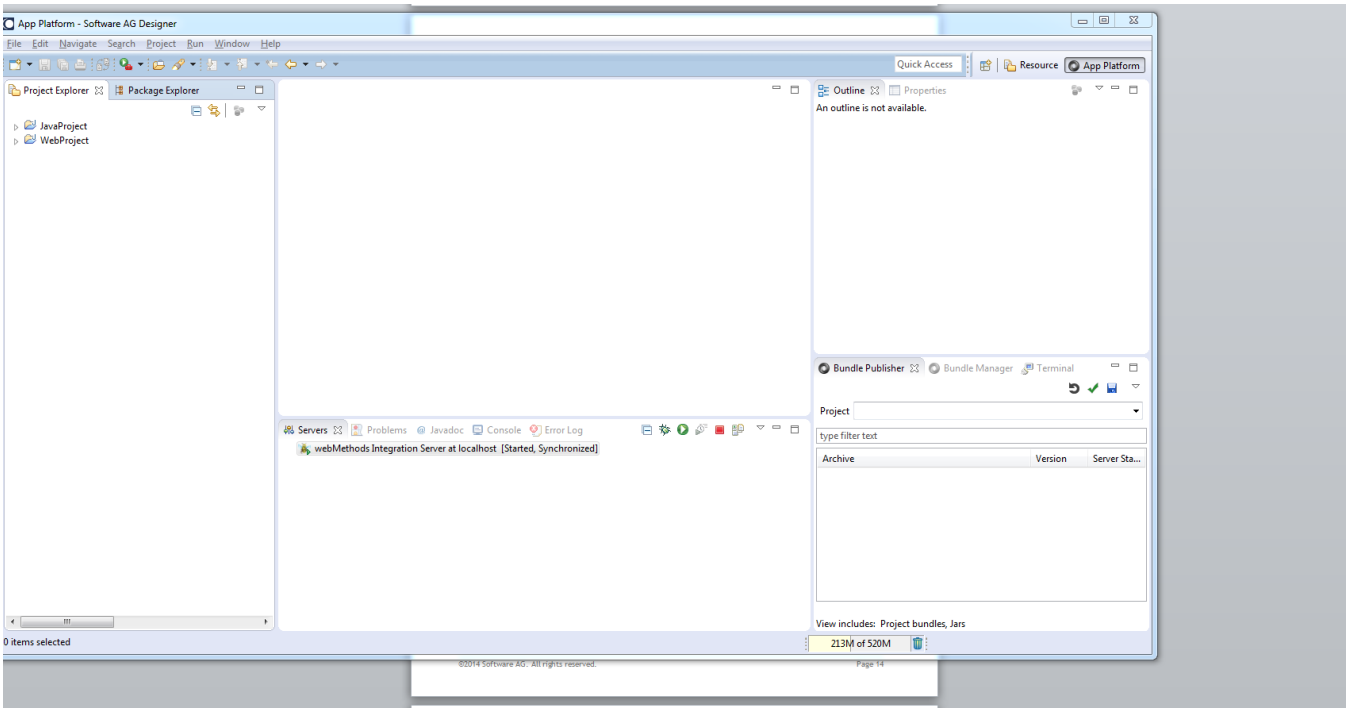
3.2 Application Platform Perspective

Eclipse uses perspectives to organize a set of editors and views in the workbench to assist with specific development tasks. The Application Platform provides a custom perspective to assist with project development. The perspective contains a collection of default views which are described in the sections to follow. Many of the views are core Eclipse components, so in these cases, please refer to Eclipse documentation for more details.

Note: Users may customize this perspective using core Eclipse tooling. It is easy to return the perspective back to its original defaults by right-clicking on the "App Platform" button in the upper right and select "Reset".

3.2.1 Views

The perspective default layout contains the following views:



3.2.1.1 Project Explorer

Use this view to access projects. Consult the Workbench User Guide in Designer's Help Contents for more details.

3.2.1.2 Package Explorer

Use this view to access projects. Consult the Workbench User Guide in Designer's Help Contents for more details.

3.2.1.3 Main Code Editor

Use this view to edit selected resources. Consult the Workbench User Guide in Designer's Help Contents for more details.

3.2.1.4 Outline

Use this view to display an outline of the current resource in the code editor window. Consult the Workbench User Guide in Designer's Help Contents for more details.

Note: Not every resource will have content in the outline view.

3.2.1.5 Properties

Use this view to display properties of the current resource in the code editor window. Consult the Workbench User Guide in Designer's Help Contents for more details.

Note: Not every resource will have content in the properties view.

3.2.1.6 Servers

Use this view to start and stop the server or to publish (or unpublish) Application Platform projects.

Add a picture here.

3.2.1.7 Problems

Use this view to resolve errors such as compilation errors in project source files. Consult the Workbench User Guide in Designer's Help Contents for more details.

3.2.1.8 Java Doc

Use this view to display Javadoc source documentation for the selected Java source file in the code editor window. Consult the Workbench User Guide in Designer's Help Contents for more details.

3.2.1.9 Console

Use this view to display content written to the system IO streams (i.e. stdout & stderr) or process input (i.e. stdin). Consult the Designer Workbench User Guide for more details.

3.2.1.10 Error Log

Use this view to display messages written to the Designer's log file located in this location: {workspace directory}/.metadata/.log Consult the Workbench User Guide in Designer's Help Contents for more details.

3.2.1.11 Bundle Publisher

Use this view to publish or un-publish additional bundles to the server. See the Bundle Publisher section for more details.

Note: Application Platform product documentation uses publish and unpublish as a means to clearly distinguish between deployment activities performed in a development environment using Designer. References to deploy and undeploy implies project deployment performed outside of Designer using ABE and Deployer. For more details regarding deployment via Deployer, please see the Project Deployment section.

3.2.1.12 Bundle Manager

Use this view to create or delete wrapper bundles that wrap non-OSGi jars. See the Bundle Manager section for more details.

3.2.1.13 Terminal

Use this view to open a telnet connection to the server profile's OSGi console.

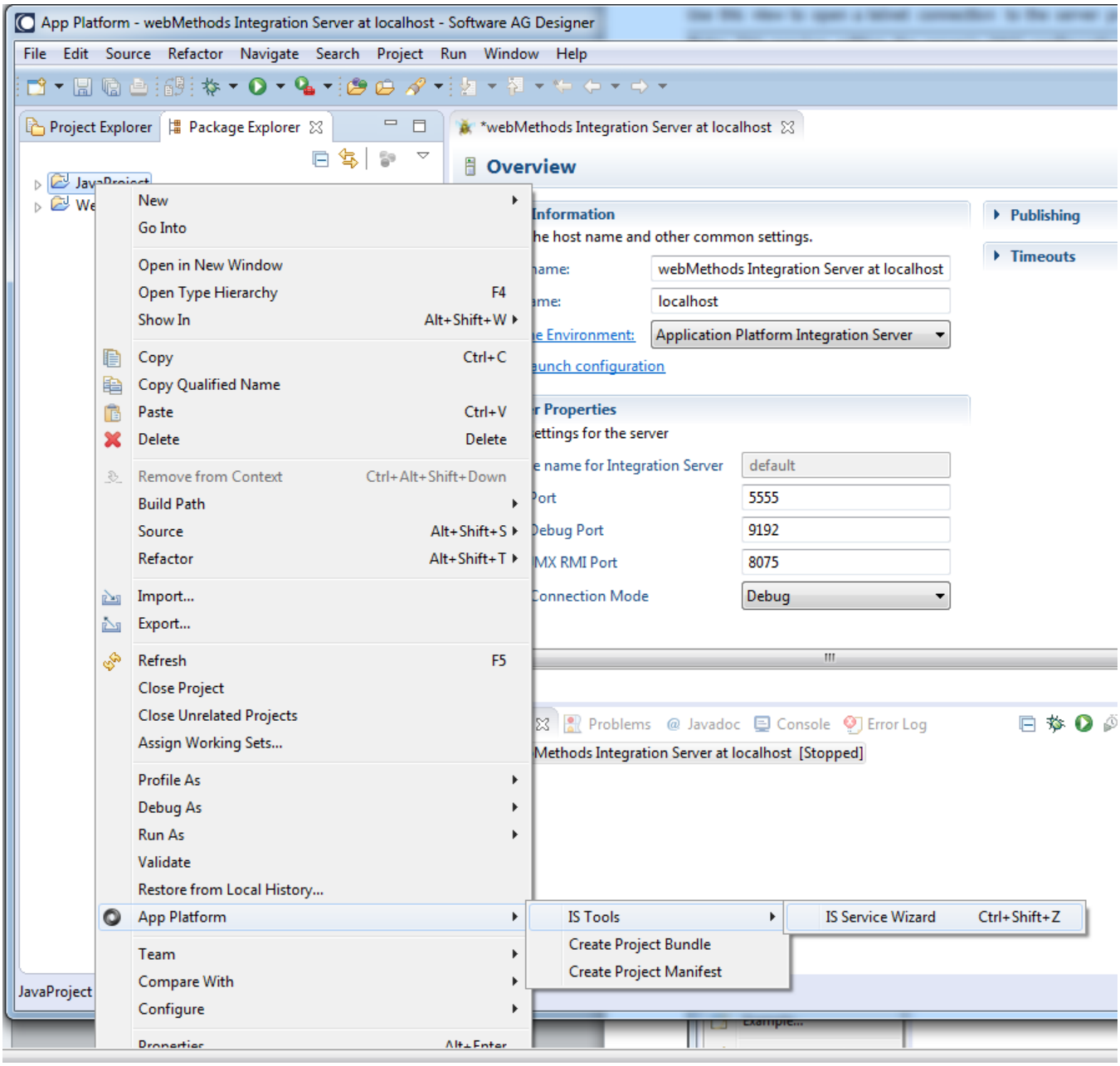
Note: This requires editing the server's OSGi configuration and restarting the server. Please see OSGi Console section for the steps needed to enable an OSGi console in a server instance.

3.2.2 Context Menu

Application Platform contributes its own context menu for executing wizards and utilities. The tools are divided into two categories:

- Core Tools – available regardless of the server product
- Product-Specific Tools - only available for a specified server product.

The following screenshot illustrates the distinction between core and product-specific. Note the product-specific sub-menu called "IS Tools". It contains an Integration Server-specific wizard called "*IS Service Wizard*". Conversely, the Core tools ("*Create Project Bundle" and "Create Project Manifest"* are available for all supported server products.



Note: the *IS Tools* menu will not be included unless the Service Integration install component is selected when installing Application Platform components for Designer.

3.2.2.1 Create Project Bundle

This menu item creates an OSGi bundle for the project currently selected in the Package or Project view. The bundle will be written to an *artifacts* folder that resides in the current workspace. The location is partially based upon the project name. For example, given a project named *MyJavaProject*, a bundle will be created in

{workspace}/.metadata/.plugins/com.softwareag.ide.eclipse.pld.bundle.builder.ui/MyJavaProject/artifacts/JavaProject.jar

Note: This is a diagnostic tool that can be used to create a project bundle without defining a server configuration and publishing a project to a server. It is not required to publish bundles into a server.

3.2.2.2 *Create Project Manifest*

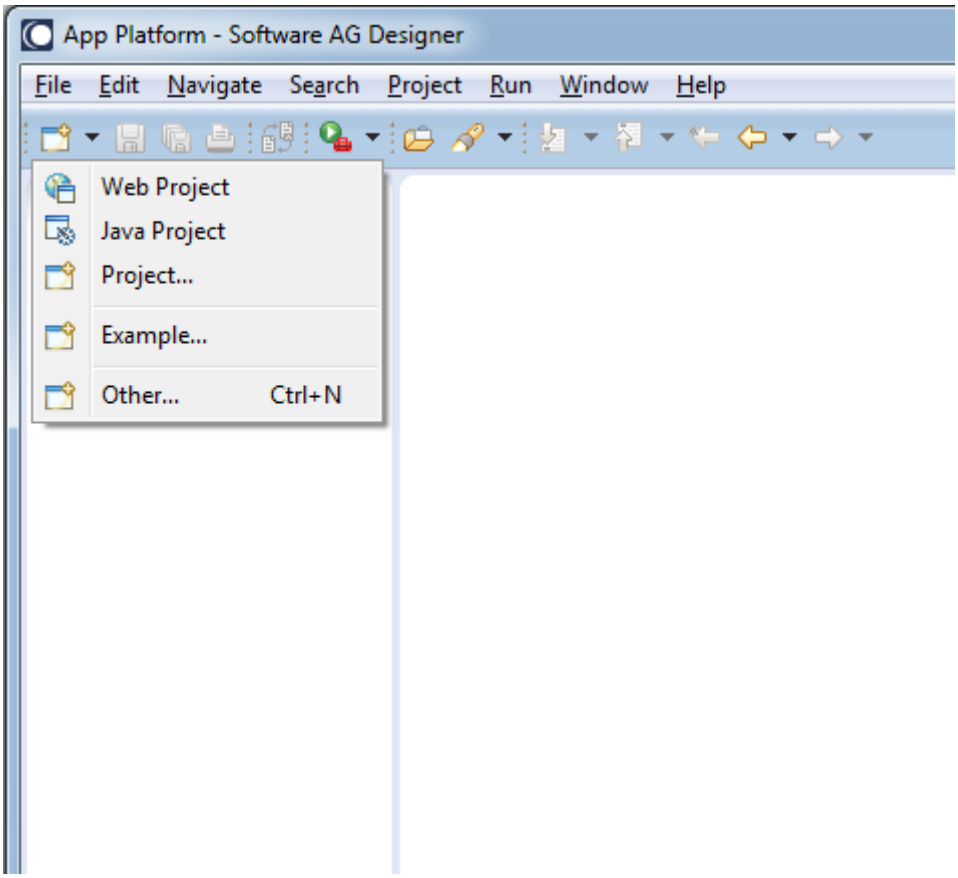
This menu item creates a MANIFEST.MF file for the selected project. The file is written to the project's "*src/main/resources/META-INF*" folder. This can be useful when the default manifest produced during bundle creation requires additional customization. See this Project Manifests section for more details.

3.3 Building a Project Using App Platform Project Wizards

The Application Platform includes two project wizards in the App Platform perspective. These wizards create projects that meet the requirements for publishing projects into the server. There are two project wizards available.

- Web Project – Required for servlet-based projects
- Java Project – Required for all other projects.

Note: Using other project wizards (e.g. the Java perspective's "Java project" wizard) will require selection of additional Application Platform project facets before it may be published to a supported server. For more details, please refer to the Configuring an Eclipse project for Application Platform section.



The Application Platform project wizards possess some common characteristics

- Eclipse WST Project facets are used to define project attributes
- An Application Platform Runtime must be selected

Consult the *Web Tools Platform Users Guide* under Designer's Help Contents more details regarding facets or server runtimes.

3.3.1 Project Facets

Application Platform project wizards utilize *project facets* to capture additional configuration needed for publishing projects into the server. A *Project Facets* wizard page containing all the project facets registered in Designer is displayed for all of the project wizards.

Each project facet may include validation to enforce any requirements it may have. For example, it is not possible to select any of the *SoftwareAG Application Platform* facets without also selecting both the *Application Platform Core and Java* facets.

Note: Each selected project facet may optionally have a wizard page to support additional configuration. Therefore, the order and number of wizard pages displayed will vary depending upon the selected project facets.

3.3.2 Application Platform Runtime

The server runtime must be selected for a project before it may be published to the server.

3.3.3 Java Project

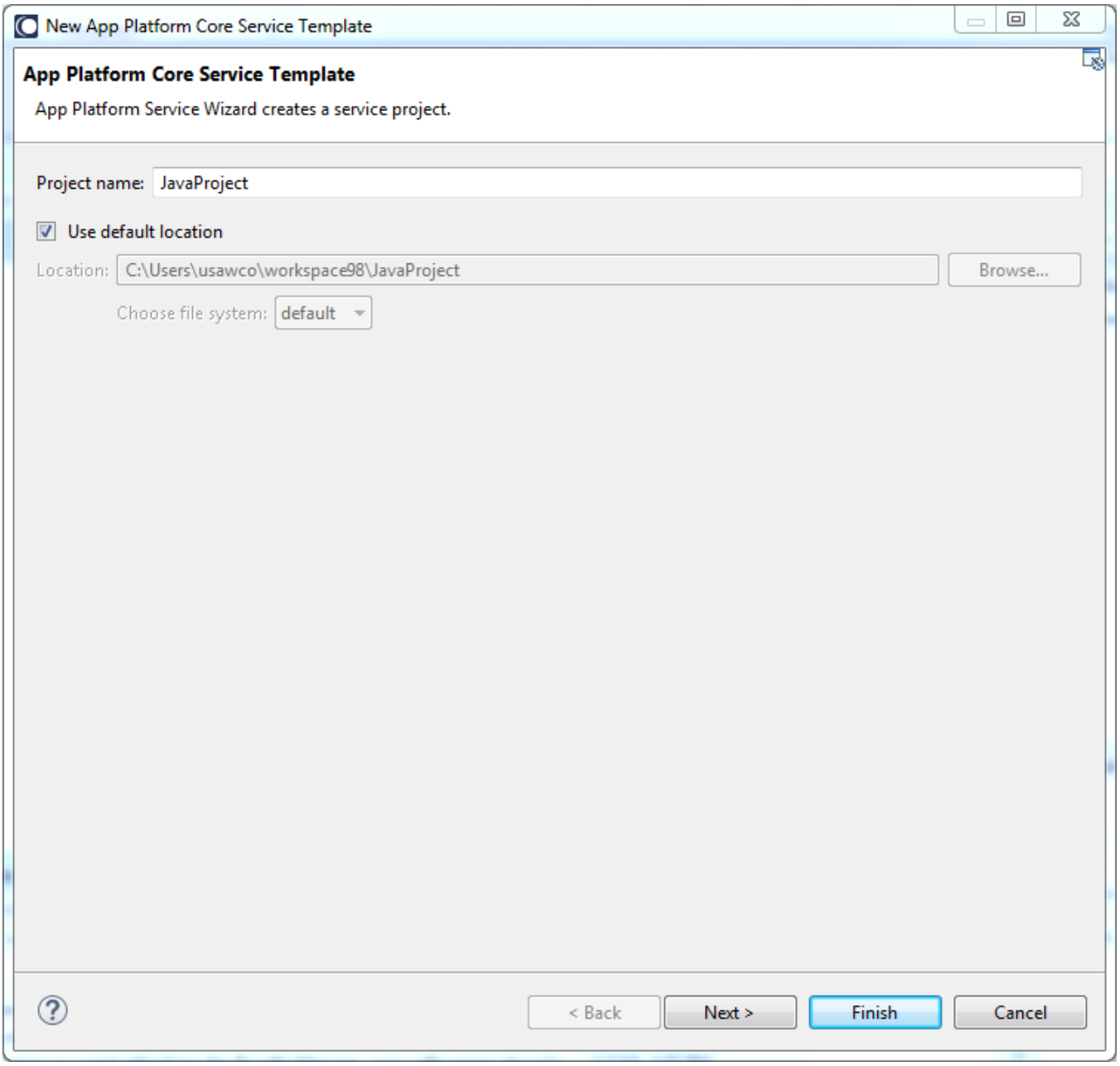
This project wizard may be used to create application components that do not require servlet support.

Note: The Java perspective also has a "Java Project" wizard that is different from this one. The Application Platform wizards are prominently displayed as the first two wizard options under the File/New menu path.

3.3.3.1 Initial Wizard Page

This is the first page for this project wizard. Enter a project name and location and press Next.

Note: It is possible to press the Finish button if enabled and the default values for the subsequent pages will be accepted.

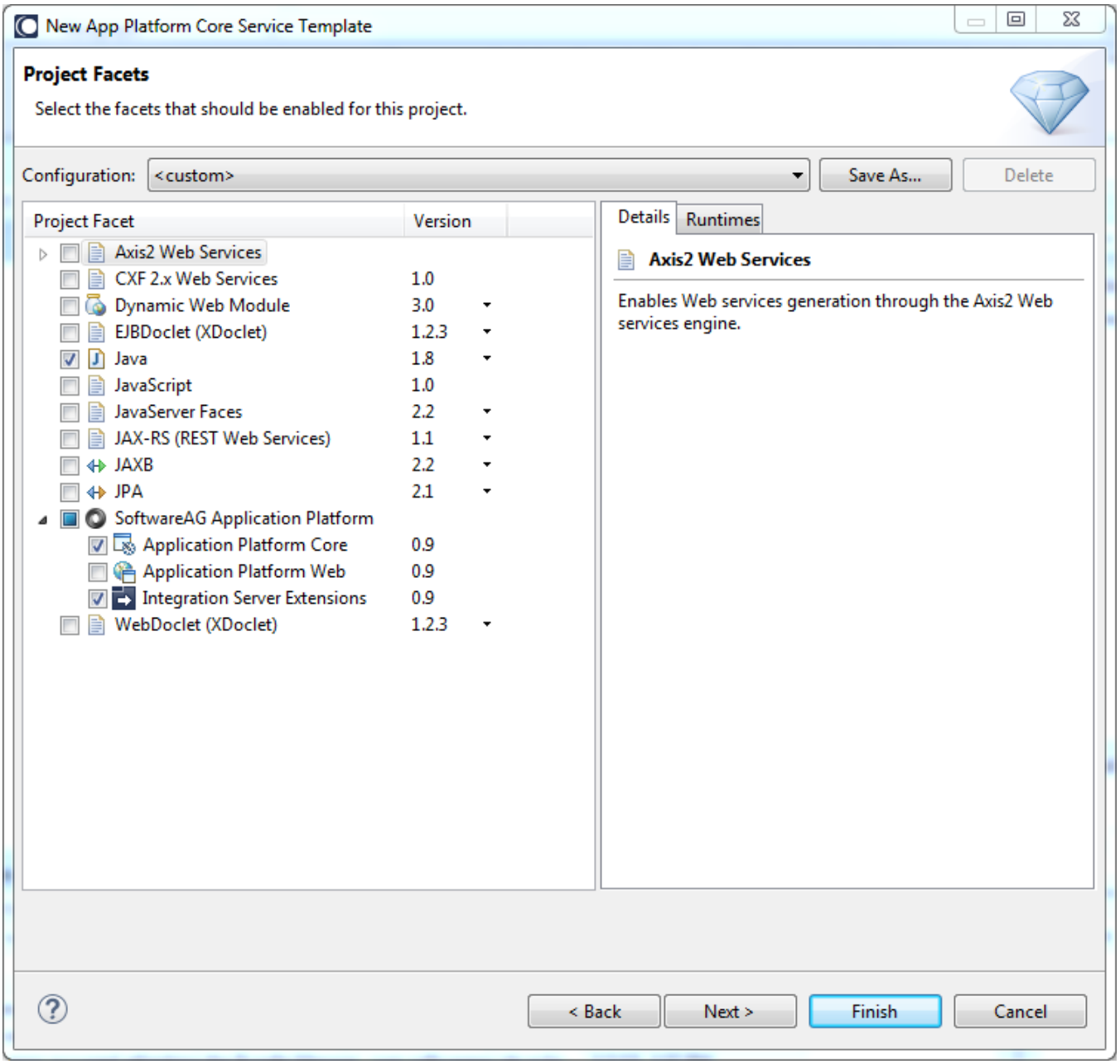


3.3.3.2 Project Facets

This page contains all the project facets which are registered in Designer. Select the necessary project facets and press the "Next" button to continue. Each project facet may include validation to enforce any requirements it may have. For example, it is not possible to select any of the "*SoftwareAG Application Platform*" group facets unless the "*Application Platform Core*" facet is included.

Each selected project facet may optionally have a wizard page to support additional configuration. The rest of this section will only discuss facet configuration for Application Platform project facets.

Note: Once a project has been created, its project facets may be modified by selecting the project in the package or project explorer and selecting *Properties* from the context menu.



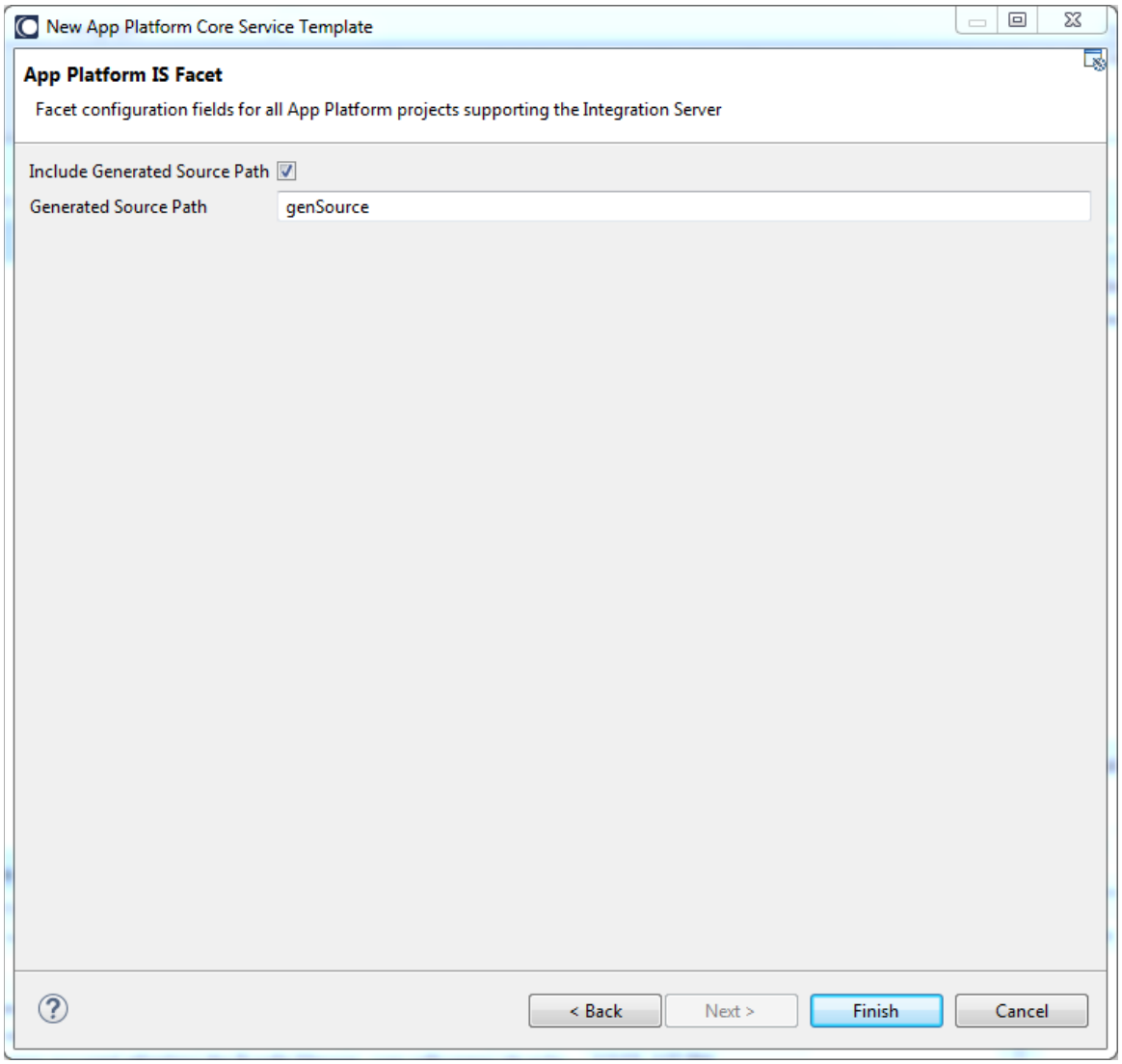
3.3.3.3 *Java Facet*

This is the Eclipse core project facet required for Java projects. The Application Platform requires the Java source directory follow the Maven 2 convention of using *src/main/java*.

Note: The Eclipse Java facet uses *src/* as its default. The Application Platform project wizard will remove this default and substitute with "*src/main/java*".

3.3.3.4 Integration Server Extensions Project Facet

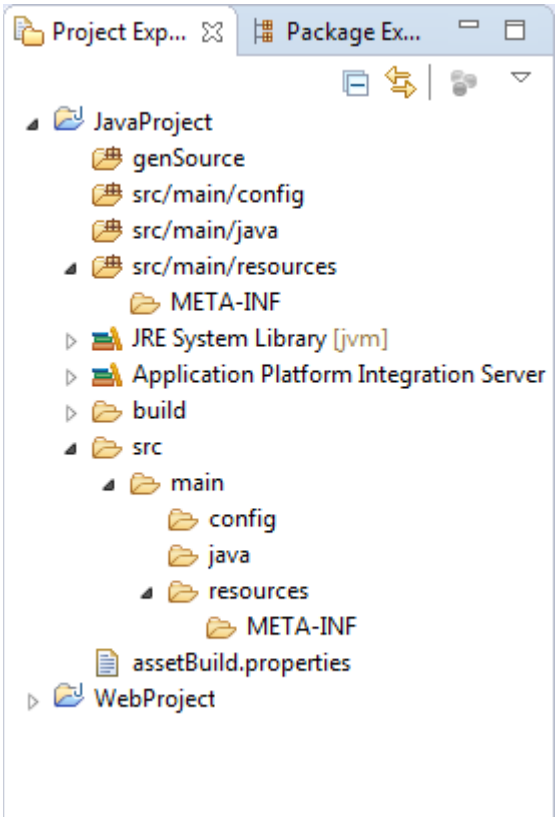
A source path must be created and added to the project's classpath when exposing Integration Server services to an Application Platform project. Checking the check box on this wizard page will ensure the directory path specified in the "Generated Source Path" text field is created for the project if it does not exist. Removing this project facet on an existing project via Project properties will remove the source directory from the project's list of source paths; however, no files or directories are deleted.



Note: The generated source directory must be *src/main/java* when deploying the project to a project environment using ABE and Deployer.

3.3.3.5 Folder Structure

After completing the project wizard, the following folder structure is created.



3.3.3.5.1 Source Folders

The source folders must follow the Maven convention (i.e. "*src/main/java*") to be compatible with the Asset Build Environment (ABE) tool. Unit test source code may be added to a *src/test/java* path.

Note: Unit tests will be included in the bundle when the project is published from Designer.

3.3.3.5.2 Config Folder

The *src/main/config* directory is a special location in the project designated to contain property files with configuration data to be passed to the server. When the bundle is published to the server, the files contained in this directory are extracted from the bundle and installed in a common directory in the server containing all the configuration files for that server. For more details, please refer to the Project Dynamic Configuration section.

3.3.3.5.3 Resource Folder

Any non-Java source files should be included in the resources directory. These files and folders defined in this location will be included in the root directory path of the bundle.

3.3.3.5.4 Lib Folder

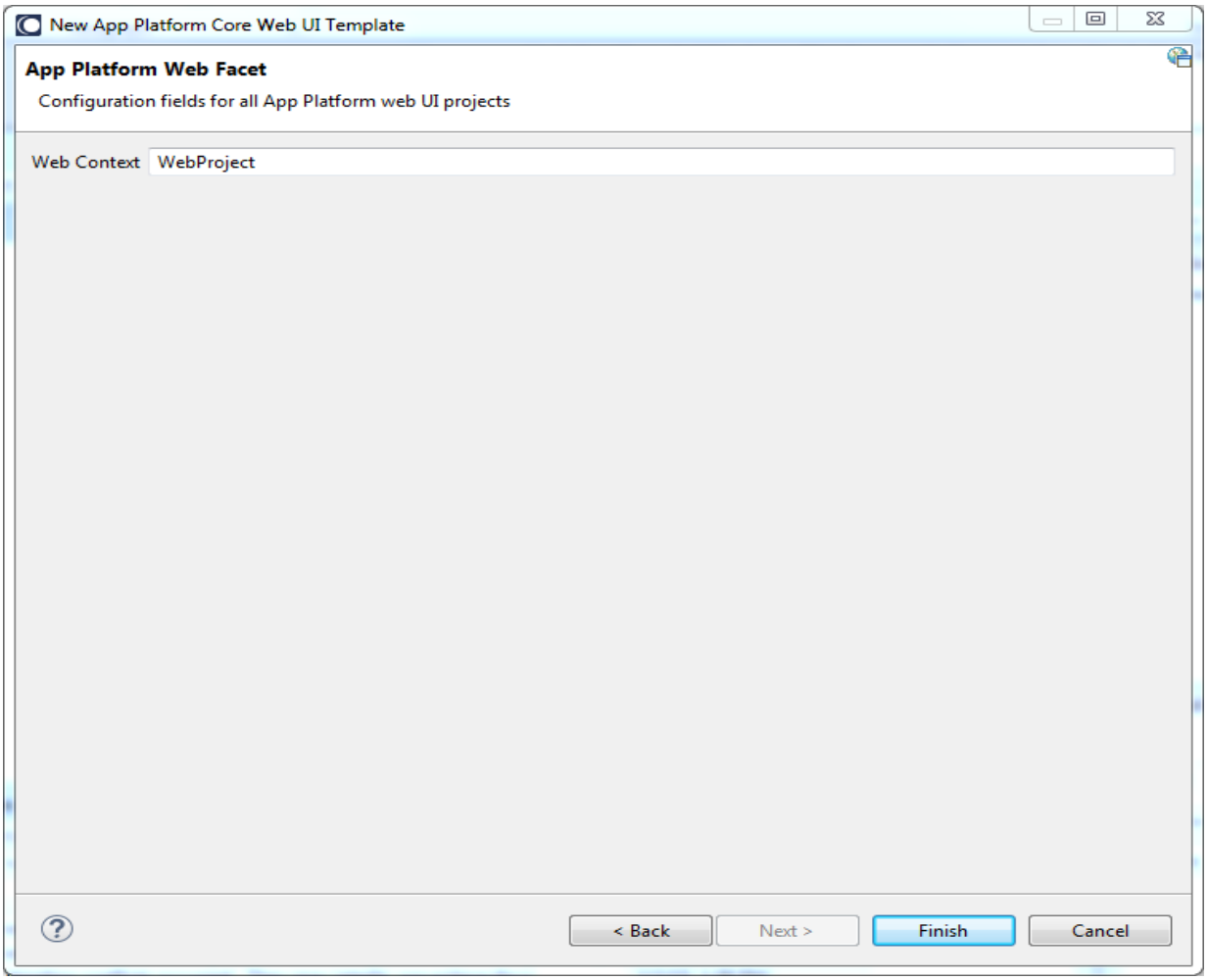
A project may include non-OSGi jar files in its classpath by including them in a lib folder. See the Including Jars in a Project section for more details.

3.3.4 Web Project

This project wizard may be used to create servlet-based application components.

3.3.4.1 Web Project Facet

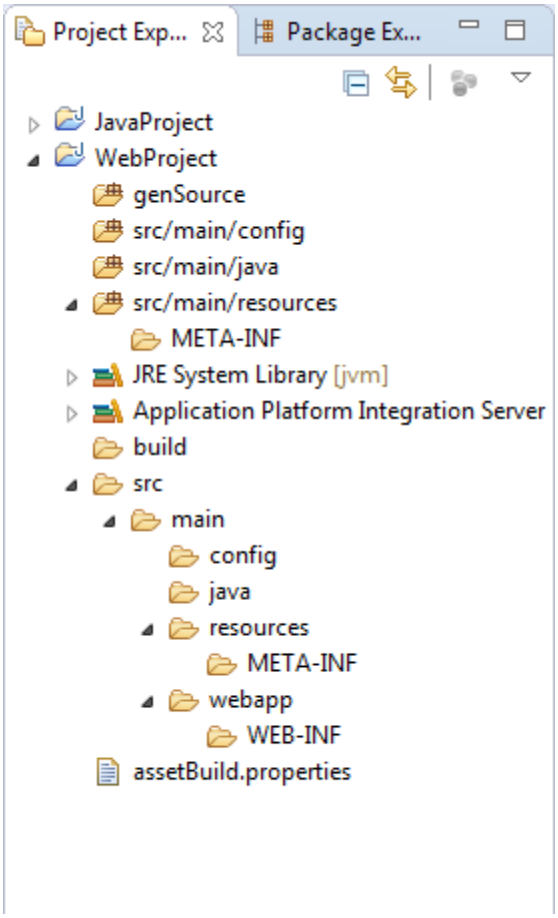
This project facet has one configuration field. The Web Context for the application defaults to the project name.



Note: When building projects using ABE intended for Deployer, the Web Context is defined by the *"Web-ContextPath:*
" OSGi manifest header property. For more details, please refer to the Project Bundle section.

3.3.4.2 *Folder Structure*

After completing the project wizard, the following folder structure is created.



In addition to the directories discussed in the prior section for Java projects, an additional folder, '*src/main/webapp/*', is created for the web-related content (e.g. HTML, JSP, JavaScript, CSS, etc.).

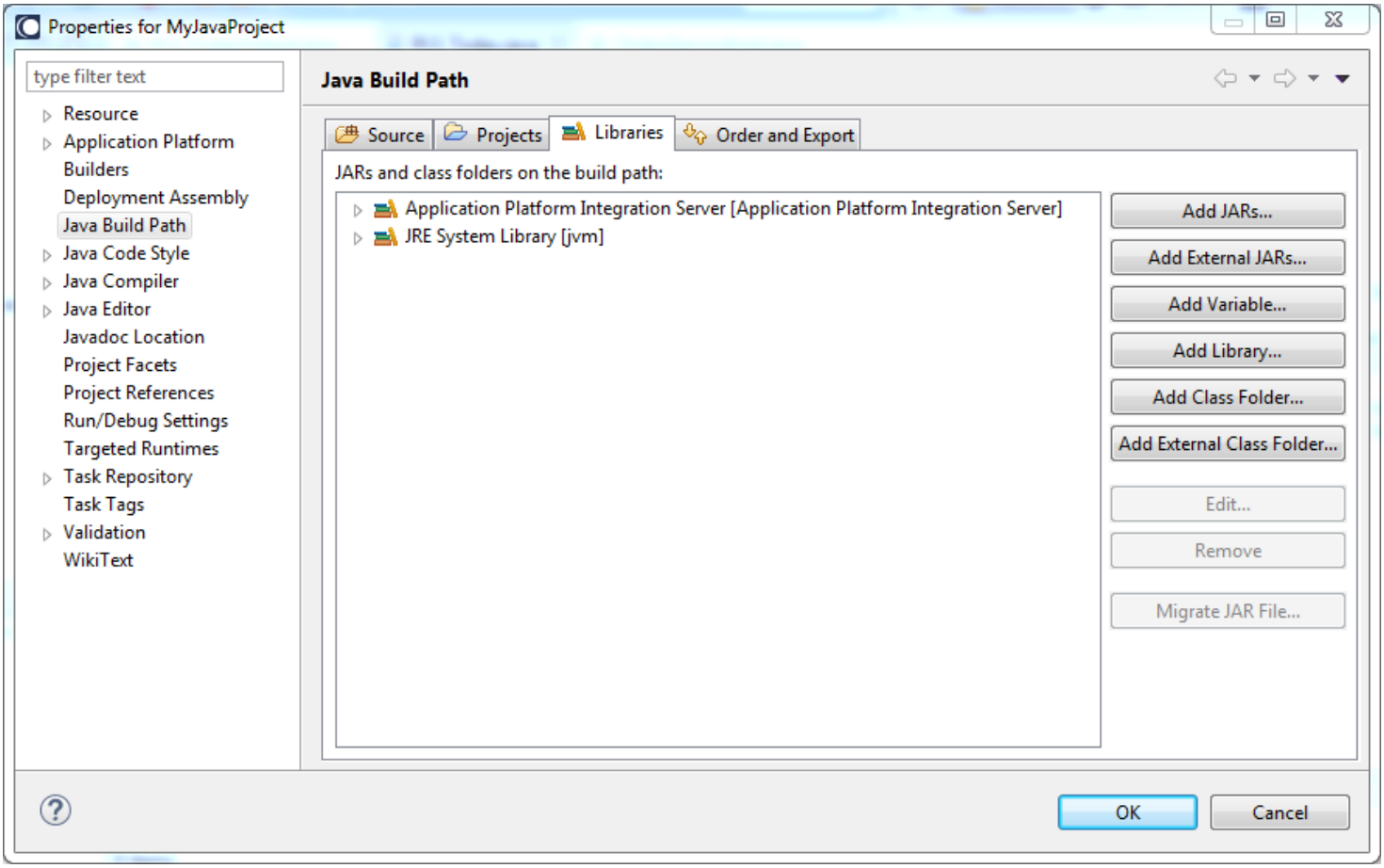
3.3.5 Classpath Containers

Classpath containers represent a collection of libraries which may be added to a project's classpath. Application Platform has two such containers. One container contains a fixed collection of product libraries (Application Platform server runtime container) while the other (Application Platform shared bundles) may be configured for each project to contain an arbitrary set of libraries.

3.3.5.1 Application Platform Server Runtime Container

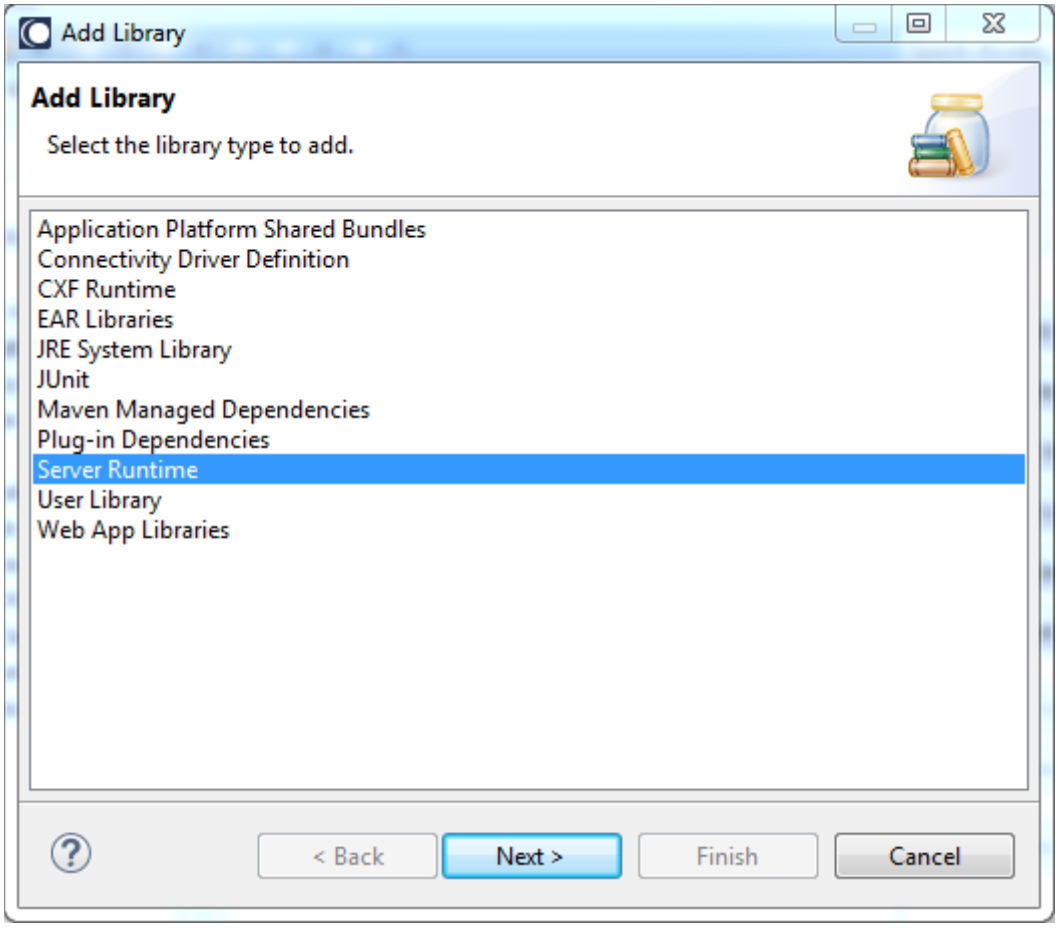
The Application Platform server container delivers a subset of the server's runtime libraries. When a project is created using one of App Platform project wizards, this container is automatically added to its classpath. Clicking on the triangle in the screenshot below will show the available server libraries. The list of libraries is pre-determined for the product and cannot be changed by users.

Application projects should take care to use these libraries rather than include duplicate versions.

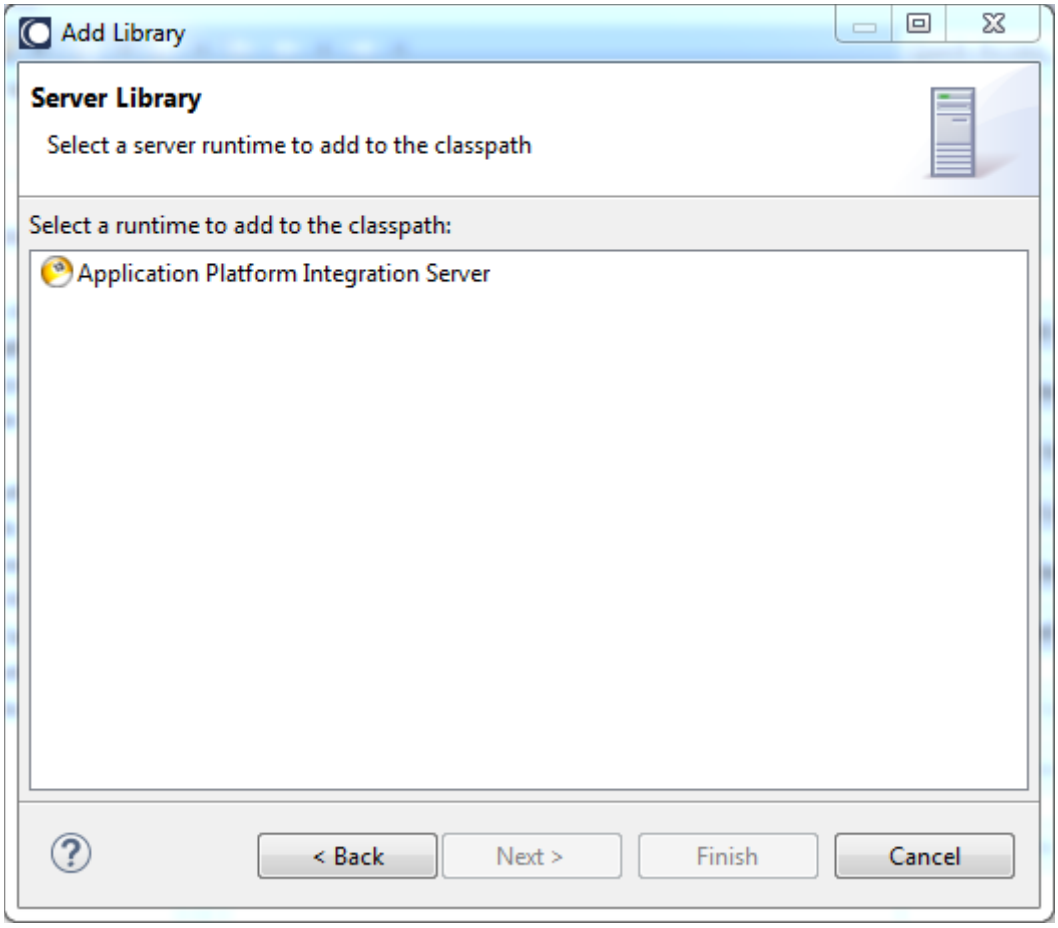


3.3.5.1.1 Adding the Application Platform Server Runtime Container

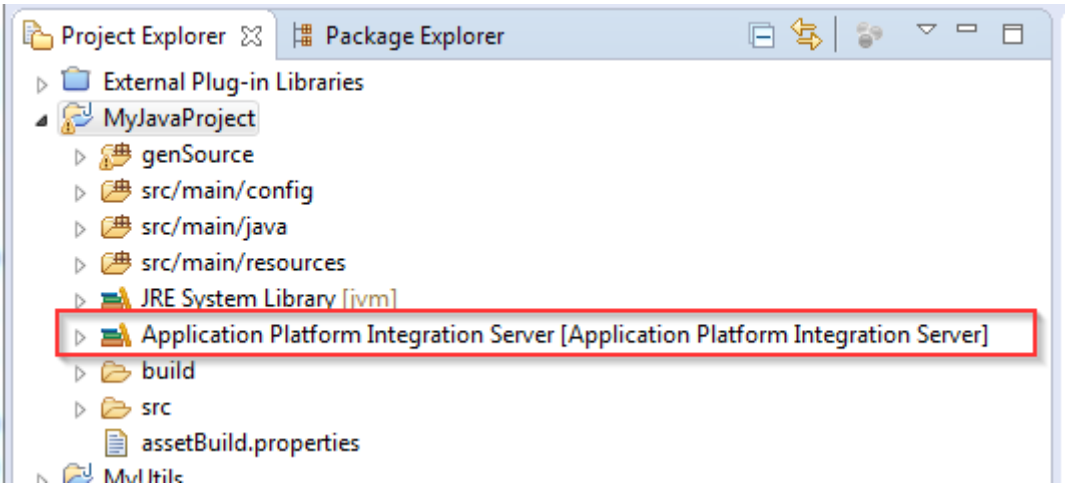
Click the *Add Library* button in the *Java build path* and select *Server Runtime*.



Next, select the Application Platform server container and click Finish.



The runtime server container should be visible on the project.



3.3.5.2 Application Platform Shared Bundles

The Application Platform shared bundles container also represents a collection of libraries that may be added to a project's classpath. These dependencies in turn can be one of the following:

- shared common jar that is not a bundle
- shared common bundles

The shared common jars can reside in some external location that is version controlled or is present in a repository library such as Artifactory or Maven Repository (Nexus).

Common jars which are not bundles can be used in two ways:

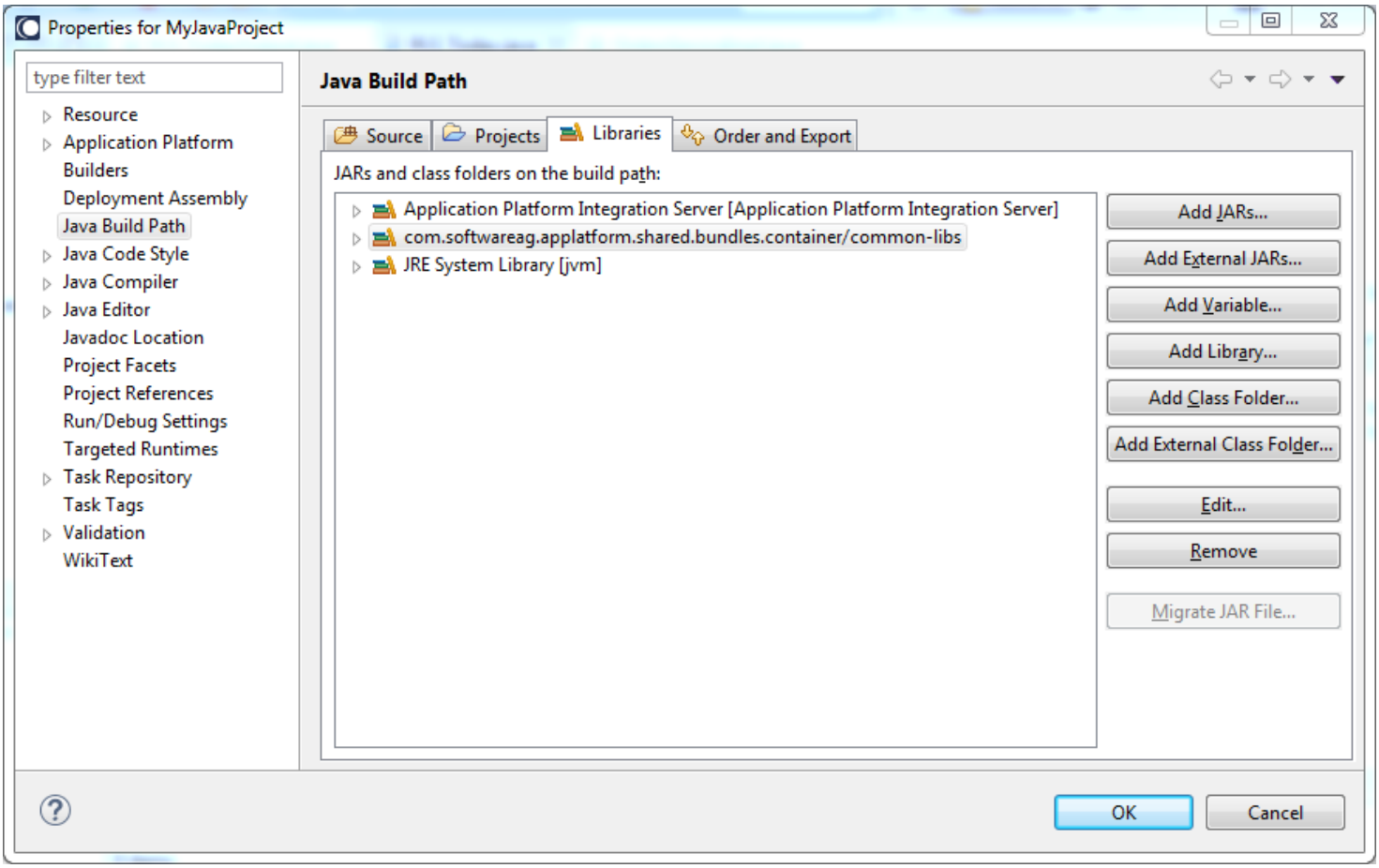
1. Included within the project as a local dependency when the project is published
2. Published as a bundle in the runtime so it can be shared with other published projects

In the 1st case, the common jars are included in the project's "lib" directory; therefore, they are duplicated across different projects during publish time.

In the 2nd case, it is wrapped as a bundle and published once into the runtime and then referred by the projects that require it. Please refer to the Managing Dependencies section for more details. Next, each project may define its own "*Application Platform Shared Bundles Container*" classpath container. This custom classpath container allows the user to specify the directory where the common bundles reside on their file system. Selecting this directory would create a new classpath library entry for the project with the bundles in it. Only valid OSGi bundles should reside in this directory; if any non-OSGi jars are present, they will not be included as part of the library entry.

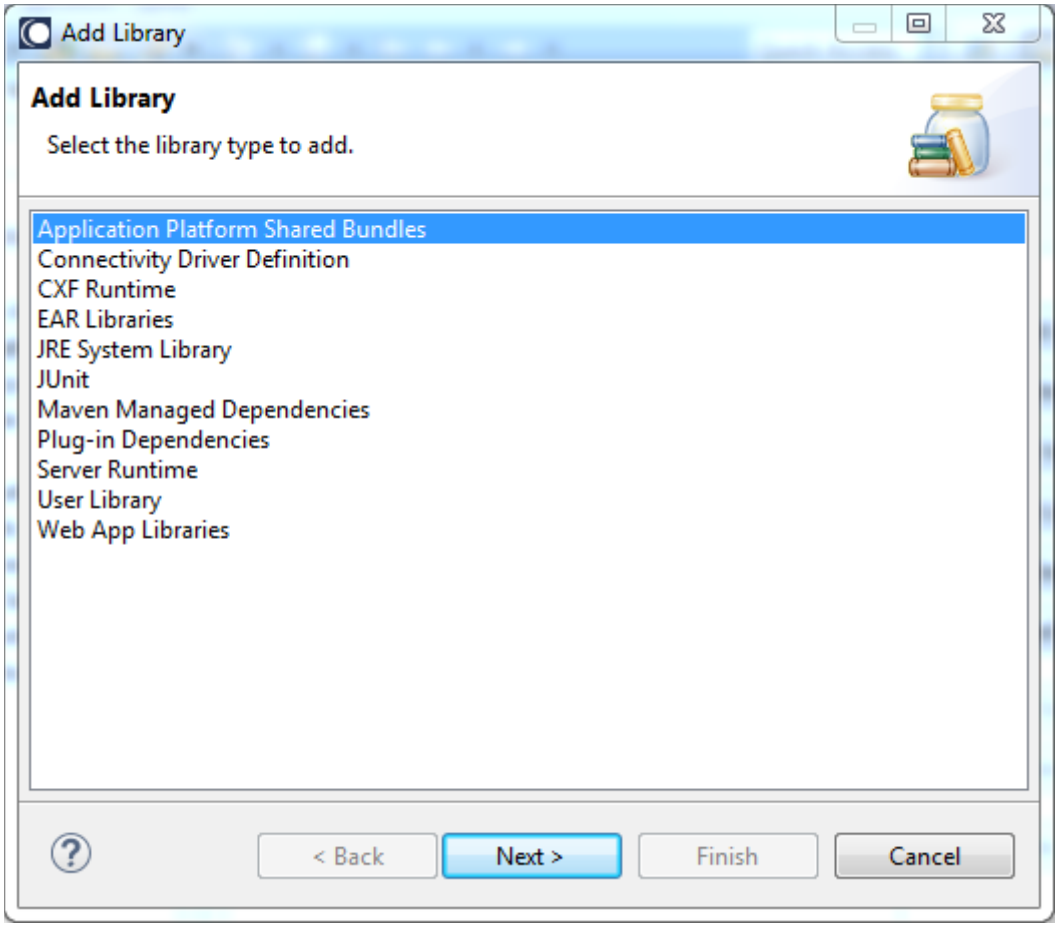
Bundles included in this fashion will not be packaged with the project and will only be used for compile dependencies and to compute the OSGi manifest's Import-Package OSGi header values when the project bundle is built. Hence, the referenced bundles must reside in the runtime before the project bundle is used. The imports will be set to be optionally required to ensure the bundle can be installed and resolved

The screenshot below shows a shared bundles container for *the /common-libs* directory path. Clicking on the triangle in the screenshot below will show the available libraries.

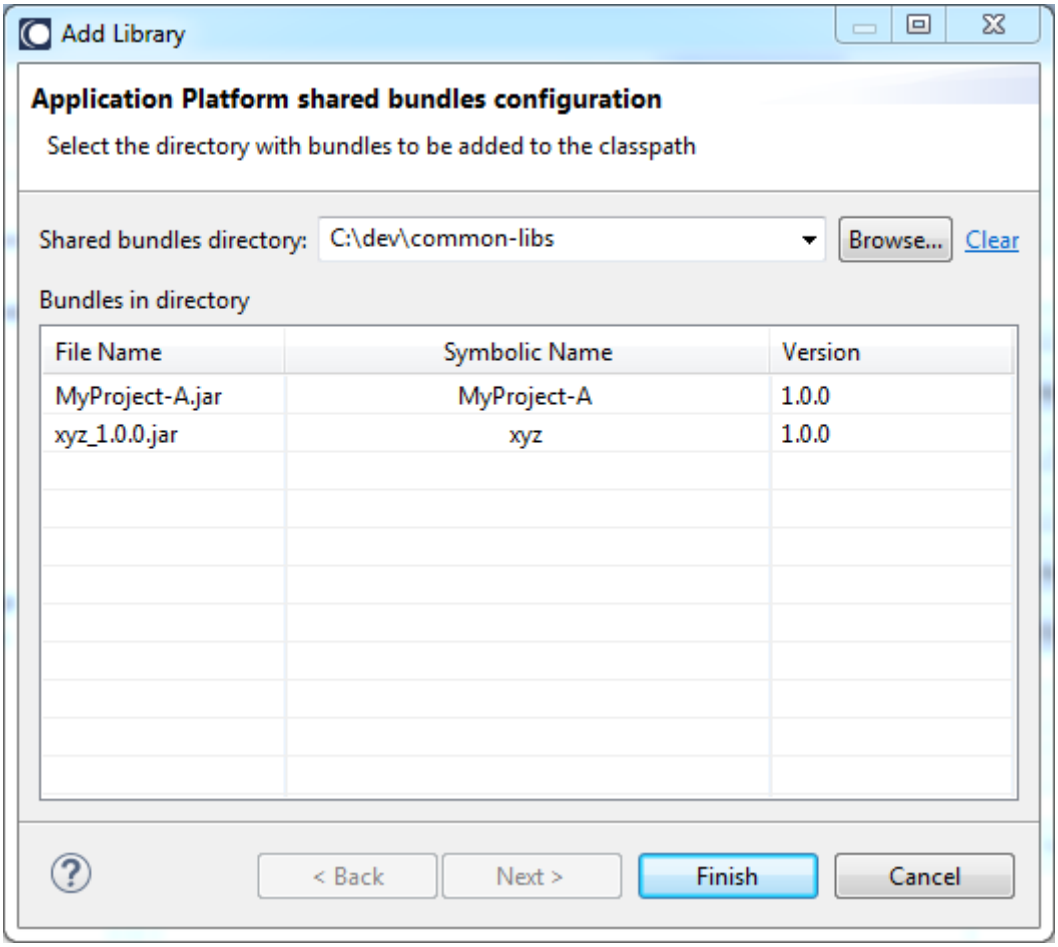


3.3.5.2.1 Adding a Shared Bundle Container

Click the *Add Library* button in the *Java build path* and select *Application Platform Shared Bundles*.

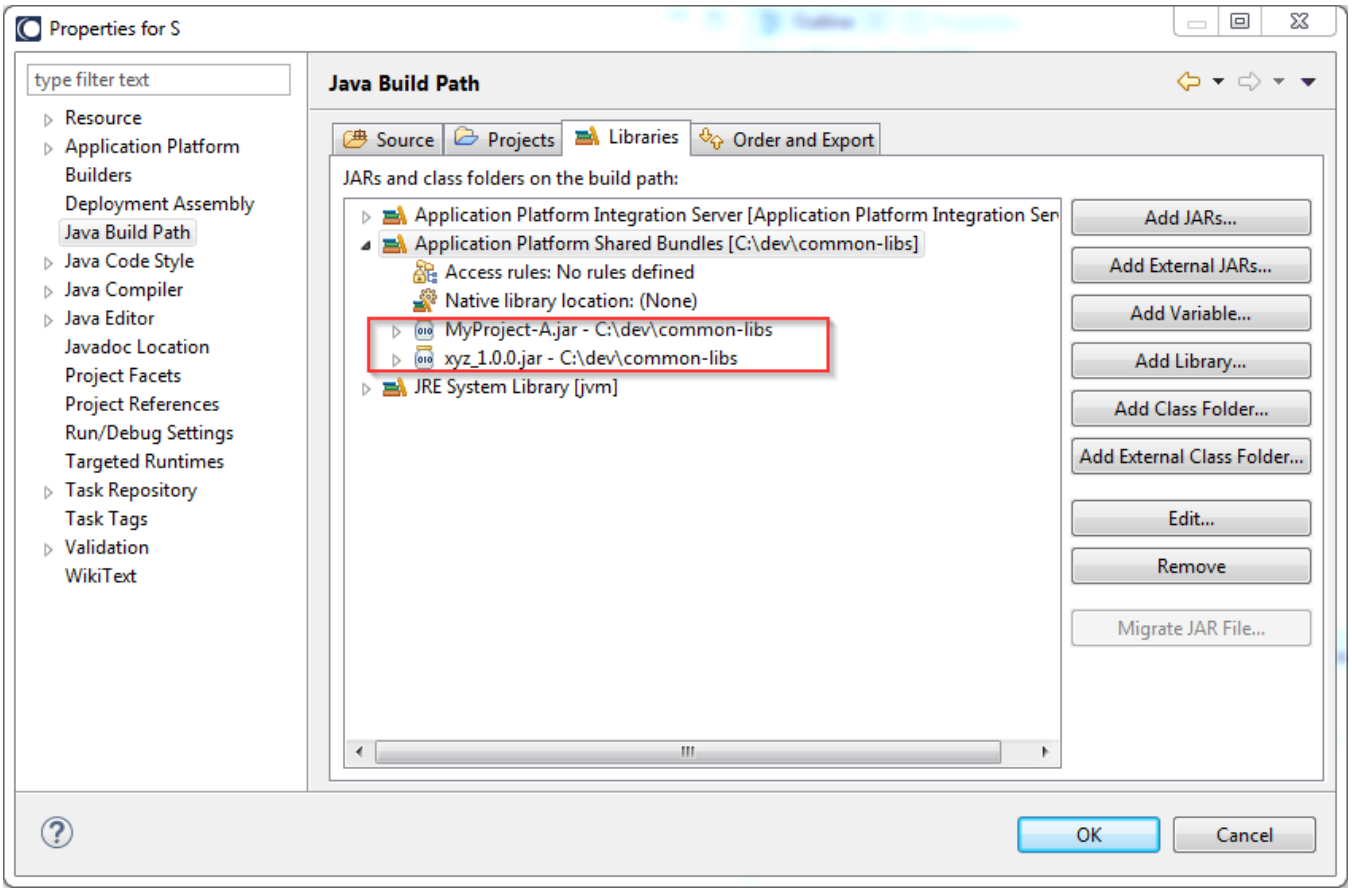


Next, Click the browse button and navigate to the directory containing the bundles.



Note: All bundles in the directory will be added to the project's classpath.

The shared bundles should be visible on the project.



Note: If bundles are added, removed, or changed in the configured directory, it may be necessary to refresh the project and perform a clean build to ensure the project's classpath is current.

3.3.6 Project Manifests

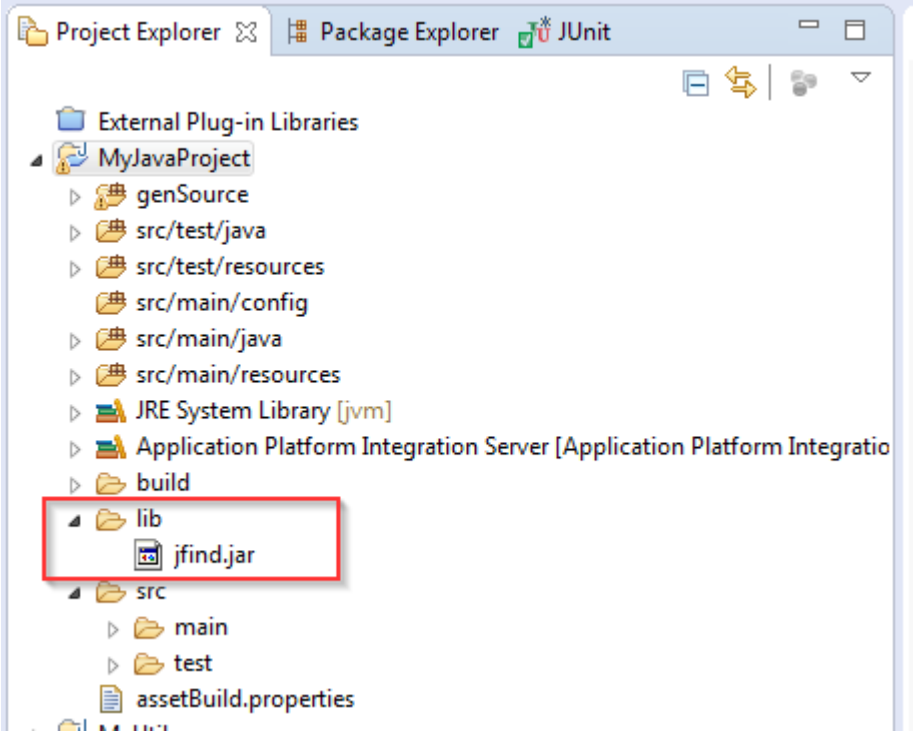
In the simplest case, when a project is published to the server, an OSGi-compliant manifest is automatically generated for the project. This manifest will contain reasonable default values for the minimal set of required OSGi headers. However, sometimes this will not be sufficient, and additional customization to the manifest will be required. In those cases, a context menu tool is available to create a manifest. To invoke this tool, please refer to Create Project Manifest section. This tool will create a *MANIFEST.MF* file in the *src/main/resources/META-INF/* directory containing default values based upon the project contents. Any customization to this file should be persisted and included in source control.

Note: Since all packages are exported by default, users are strongly encouraged to not use the same package name across projects. This can lead to a scenario where the same package and version is exported by more than one bundle. While OSGi supports this situation, it requires additional tricky manifest manipulation via split packages to avoid runtime errors.

3.3.7 Including Jars in a Project

Libraries may be included in the project's lib folder. When included in this location, the project bundle will include these additional libraries within the bundle's classpath. This provides a simple mechanism for including non-OSGi jars in a project bundle's classpath. The packages contained within these libraries will be available only to the bundle's classloader. Put another way, this means these jars will only be available to the project's classes.

For example, assume the following *jfind.jar* file is on this project's classpath.



The generated project bundle will contain the jar and the OSGi manifest will contain a Bundle-ClassPath header attribute.

Bundle-ClassPath: .,lib/jfind.jar

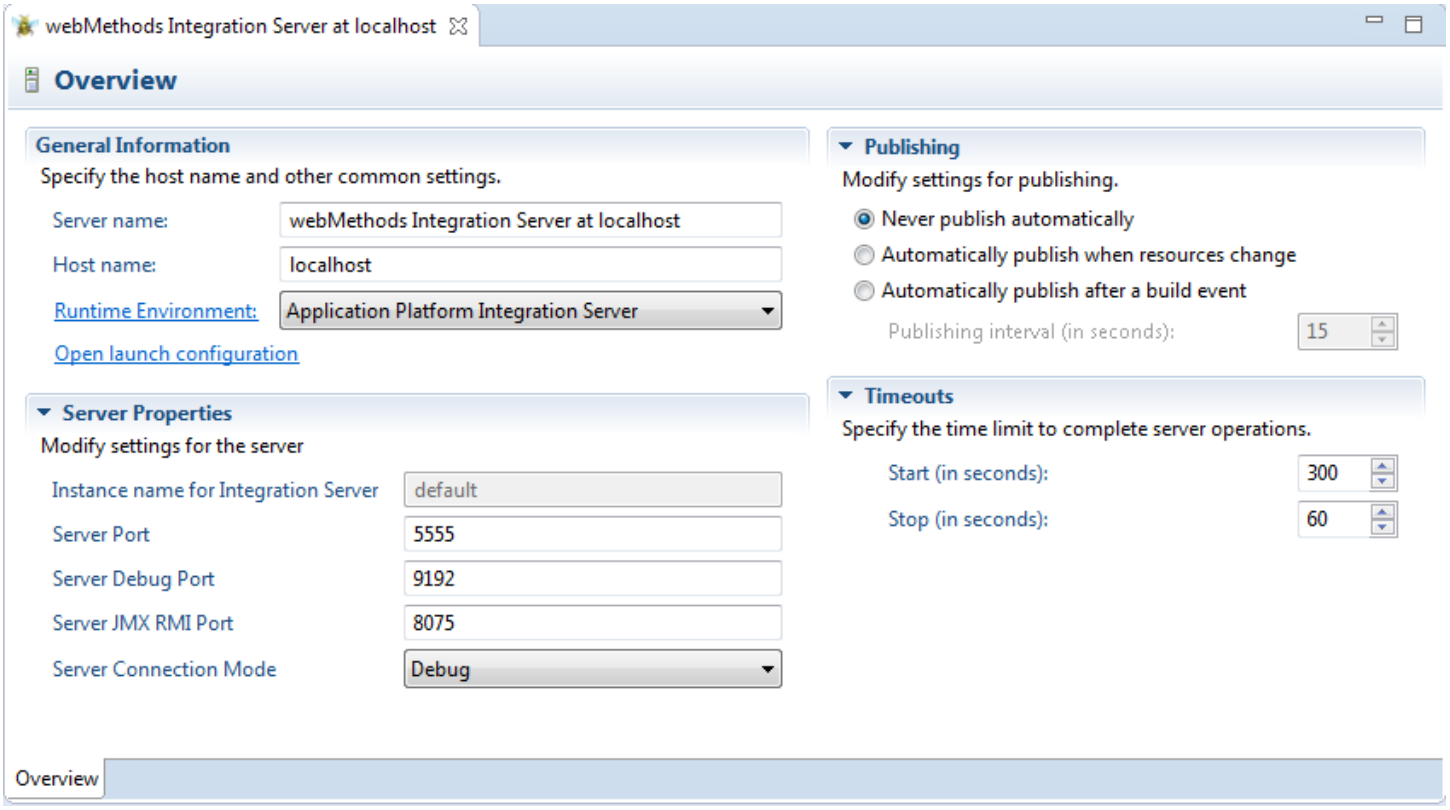
This ensures the OSGi container will include this jar's classes on the project bundle's classpath when the bundle is published. However, the jar's packages will not exported, so they are only resolved by classes inside the project bundle.

Note: For creating bundles from non-OSGi jars or publishing existing bundles to be shared across more than one project, please refer to the Managing Dependencies section.

3.4 Server Management

The Application Platform's server management is based upon the Eclipse Web Server Tools (WST) project. This screenshot shows the main configuration view for a server configuration. This configuration window is accessible by double-clicking on a server in the "*Servers*" view. The sections that follow describe these elements. The server configuration is an extension of an Eclipse feature included in the Web Tools Platform. For more details, please consult the "*Using the server tools*" section under the Web Tools Platform Guide in Designer's Help Contents.

Note: When making changes to the server configuration, unpredictable results may occur if modifying a configuration of a running server. Always stop a server before making configuration changes.



3.4.1 General Information

As indicated in the screenshot, this section includes those fields that are common to any server configuration produced by the WST framework.

3.4.1.1 Server Name

This is a free text field to provide a meaningful name for the server.

3.4.1.2 Host Name

This field contains the server's hostname.

Note: Remote servers are not supported for this release. Please do not change the localhost default value.

3.4.1.3 Runtime

The "*Runtime Environment"* link associates a server to a runtime. The runtime environment includes an absolute directory path to the Application Platform installation. This absolute path is used in conjunction with relative paths to the platform bundles which represent those server libraries which are available to a project's compile path.

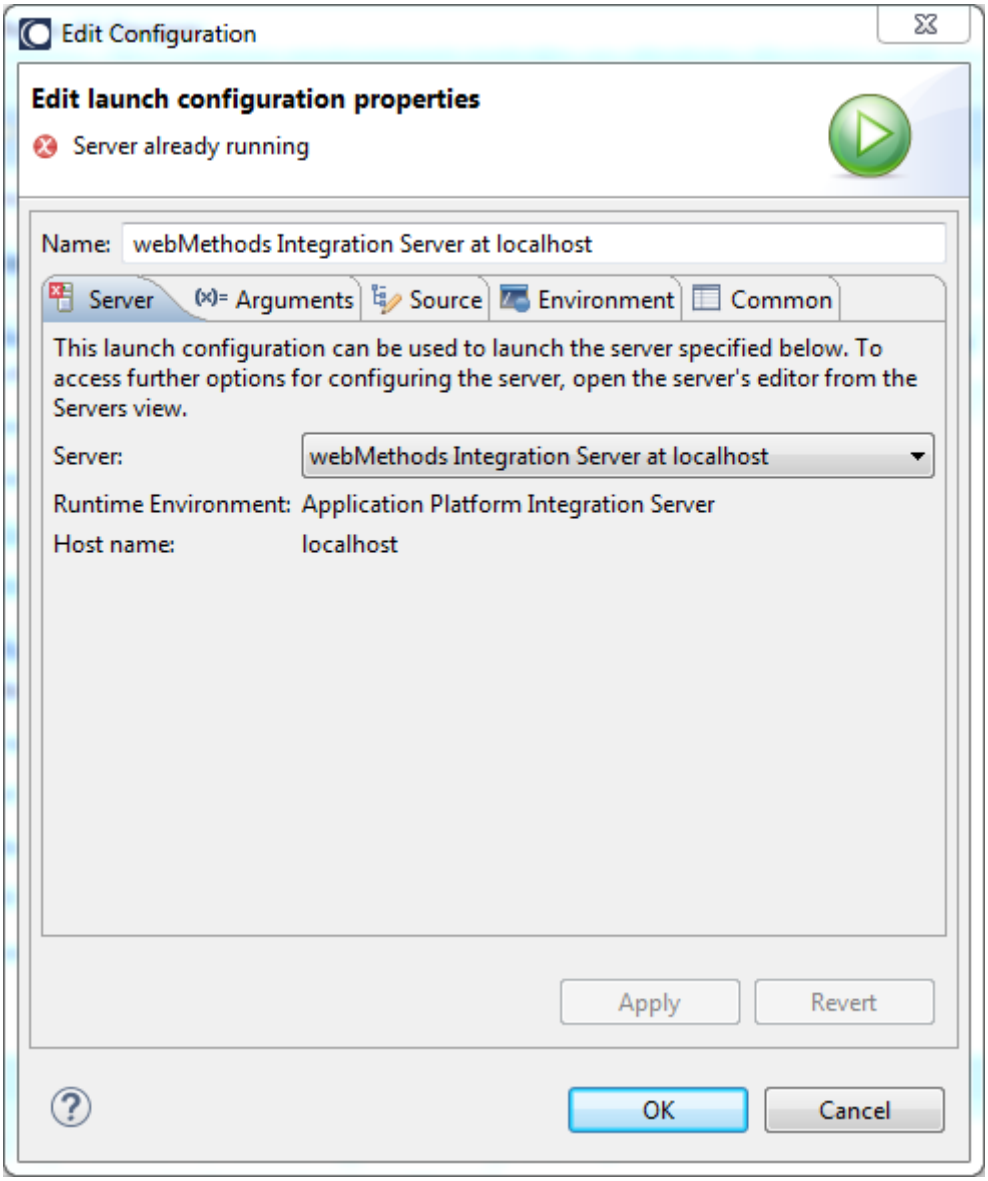
Note: The runtime configuration settings is also accessible from Window/Preferences/Server/Runtime Environment

3.4.1.4 Launch Configuration

The Open Launch Configuration link contains the configuration elements used by Designer when launching a process to execute operating system scripts used to start and stop the server. The default values found here are dependent upon the server configuration values, so they should be adequate for managing the Integration Server startup and shutdown.

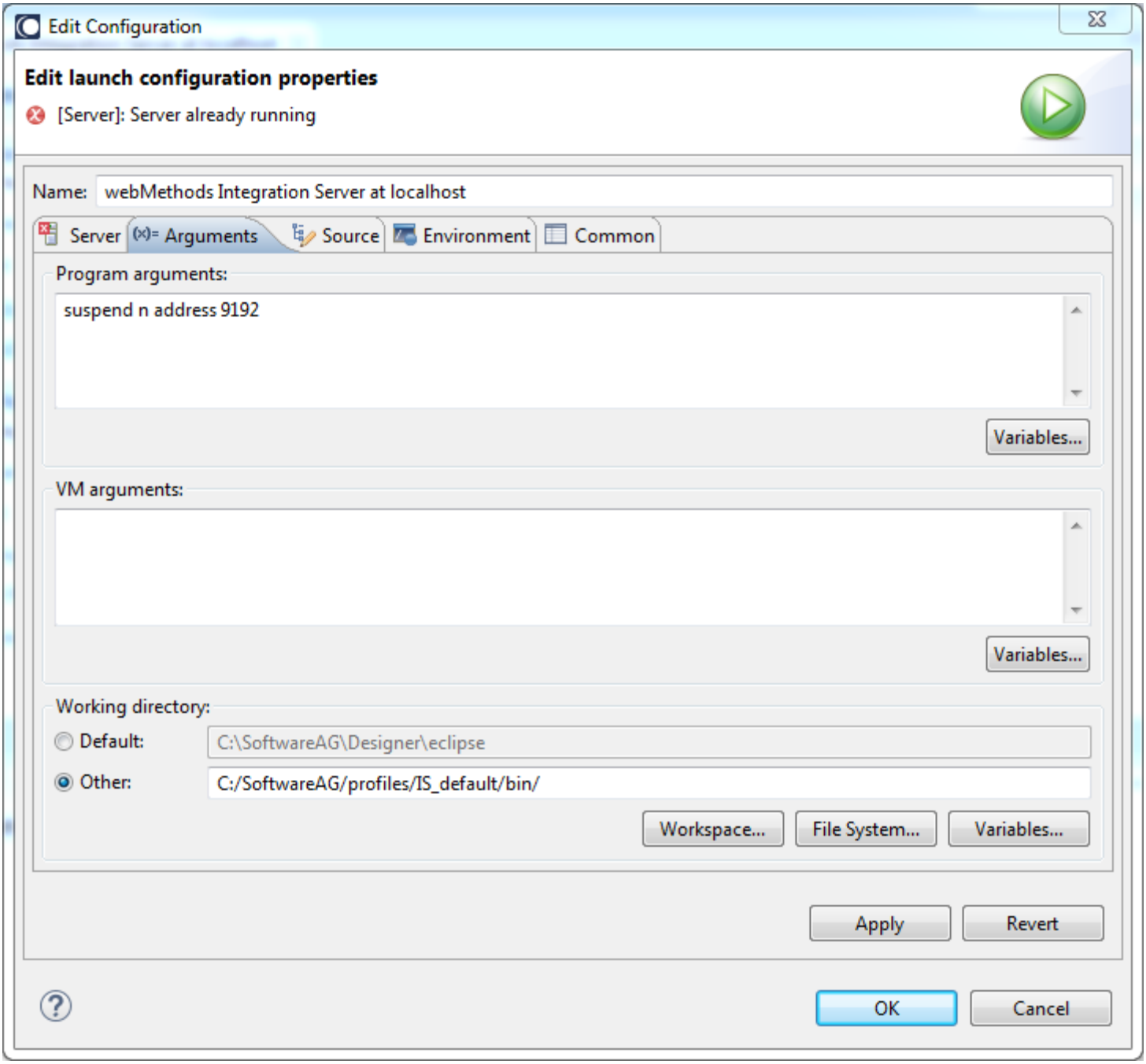
3.4.1.4.1 Server

The *Server* tab shows the currently edited server.



3.4.1.4.2 Arguments

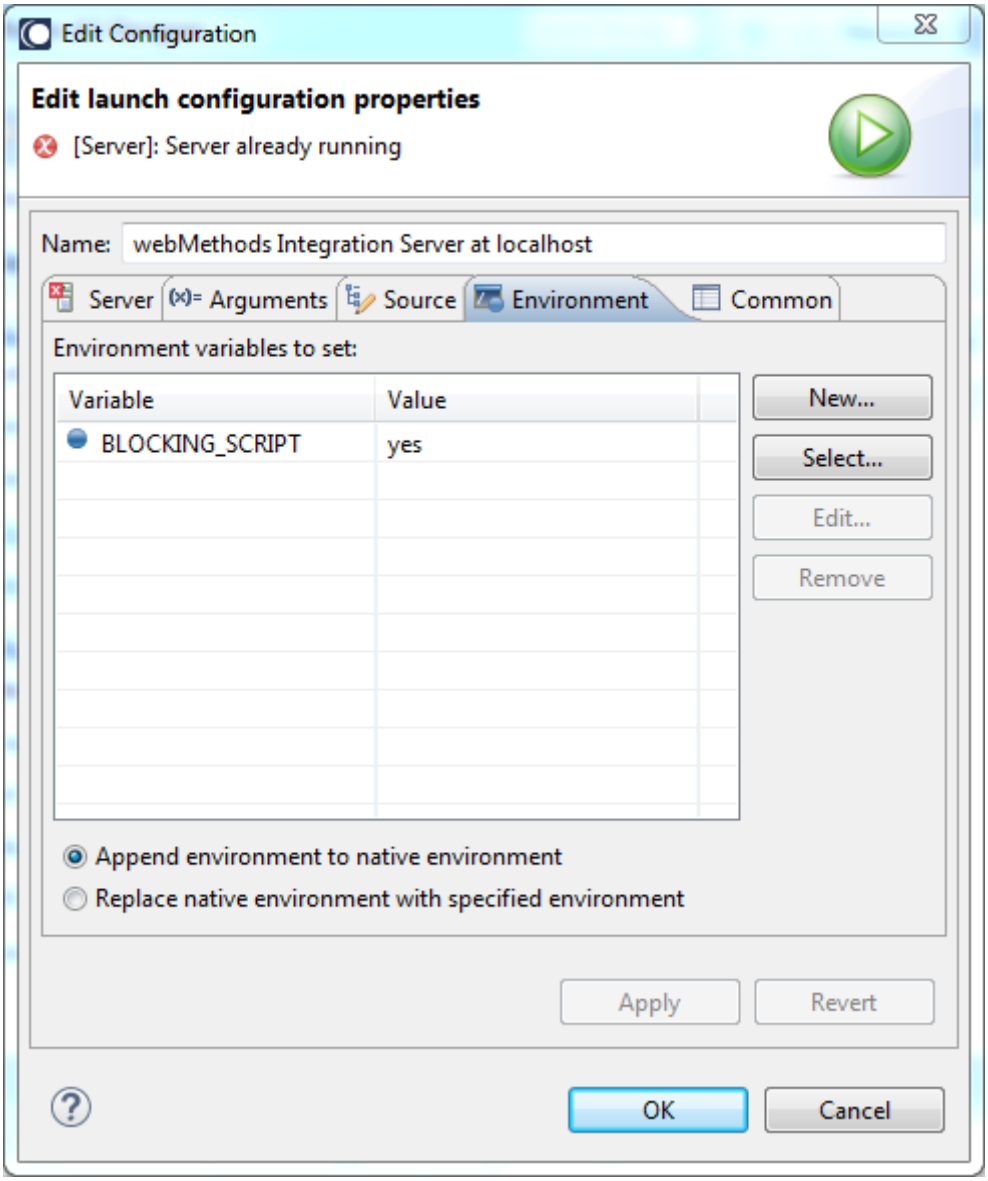
The *Arguments* tab contains arguments to be passed to the server OS script responsible for starting the server.



Important: Do not modify or delete the existing program arguments.

3.4.1.4.3 Environment

The Environment tab contains environment variables that are defined in the OS process used to launch the start script.



Important: Do not modify or delete the existing environment variables.

3.4.1.5 Publishing

This section has a radio group of options available.

3.4.1.5.1 Never Publish Automatically

This is the default setting. Users must take action to publish their project to the server. This option gives the most control for the user.

3.4.1.5.2 Automatically Publish When Resources Change

This setting will trigger an automatic project publish 15 seconds after a resource in the project has changed. The time interval is configurable as well.

Note: Using this option can be very resource intensive.

3.4.1.5.3 Automatically Publish After a Build Event

This setting will trigger an automatic project publish after project build event – i.e. Clean, full or incremental project builds.

Note: Using this option can be very resource intensive by.

3.4.1.6 Timeouts

There are two timeouts for a server configuration.

3.4.1.6.1 Server Startup

This timeout determines how long Designer will wait for a server to start before assuming failure. The default is 300 seconds. An error dialog is presented to the user if the timeout is exceeded. See the Start section below for more details.

3.4.1.6.2 Server Shutdown

This timeout determines how long Designer will wait for a server to shut down before assuming failure. The default is 60 seconds. A *Terminate Server* dialog is presented to the user after this time out has elapsed. See the Stop section below for more details.

3.4.2 Server Properties

The properties that follow are specific for the Integration Server.

3.4.2.1 Integration Server Instance Name

This field matches the instance name of the Integration Server. The default value is *default*.

Important: Only the 'default' instance is supported for this release. Do not delete the "*IS_default*" profile defined with the product installation.

3.4.2.2 Server Port

This is the HTTP port for the configured Integration Server. This port is necessary to verify the server startup sequence. The default value is 5555.

3.4.2.3 Server Debug Port

This is the JPDA debugger port configured for the Integration Server's JVM. This port value is passed to the startup scripts when the Integration Server is started by Designer. During the server startup sequence, if Designer cannot connect to the configured debugger port, the server will still start; however, any breakpoints will be ignored. The default value is 9191.

3.4.2.4 Server JMX Port

This field contains the JMX port used to execute a service in the server that supports publishing bundles into the OSGi container. The default port is 8075 which matches the configured port for the server. This port number is configured in the server by a property file found in *${sag.install.dir}/profiles/IS_default/configuration/com.softwareag.platform.config.propsloader/* directory. Please refer to document guides in the Server section for more details.

Note: If the port is in use while installing Application Platform, the port number may change in the server's configuration. If you are uncertain of the server's state, use a OS utility to see if the JMX port is in LISTEN mode.

3.4.2.5 Server Connection Mode

The *server connection mode* property determines how the Servers view gets synchronized to the state of the external servers when the IDE is started or when the server is stopped or started from outside of Designer. There are three possible states for this field.

- Debug
- No Action
- Run

Note: Since the server states are synchronized via a polling mechanism, there may be a small delay until the Servers view is updated.

3.4.2.5.1 Debug

With this setting selected, Designer will automatically synchronize the state of the server in the Servers view to "*Debugging*". This is the default value when creating a server configuration. For example, if Designer is stopped and restarted, the server instance in the "Servers" view will automatically transition to "*Debugging*".

3.4.2.5.2 No Action

With this setting selected, the state of the server in Designer is not synchronized to the server. If Designer is started and a server is running, its Servers view will indicate the server is "Stopped". In this case, the user must execute the Start or Debug action in the Servers view. Additionally, if the

server's state changes while Designer is launched, e.g. the server is stopped using the "*Stop Servers*" menu option, Designer's Server view will still indicate the server is "*Started*".

3.4.2.5.3 Run

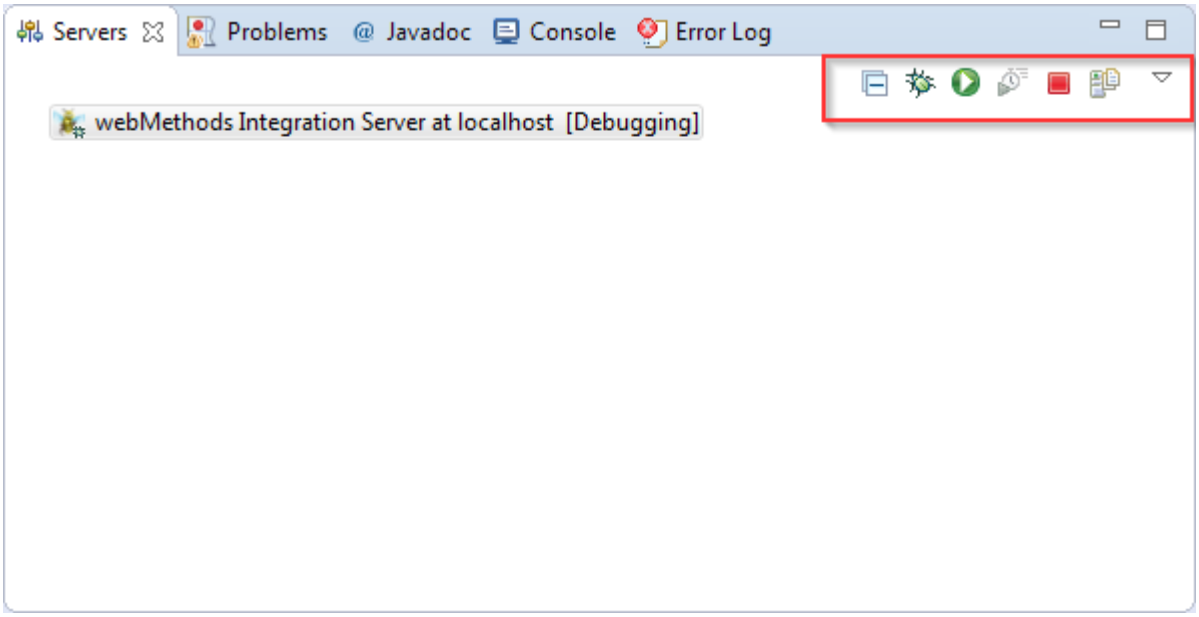
With this setting selected, Designer will automatically synchronize the server state to "*Started*". It is not possible to remotely debug applications while Designer is connected to the server in this state.

3.4.3 Server Operations

The Servers View is an Eclipse WST component customized to fulfill the needs of Software AG servers. The following operations are possible from the servers view. A view-specific toolbar contains actions for managing the server state. These actions include the following:

- Starting the server
- Stopping the server
- Debugging the server
- Publishing or un-publishing projects

The screenshot below shows the toolbar actions available in this view. The same actions are also available on the right-click context menu when selecting server instance.



Note: The toolbar action "*Starting the server in profile mode*" is not supported.

3.4.3.1 Start

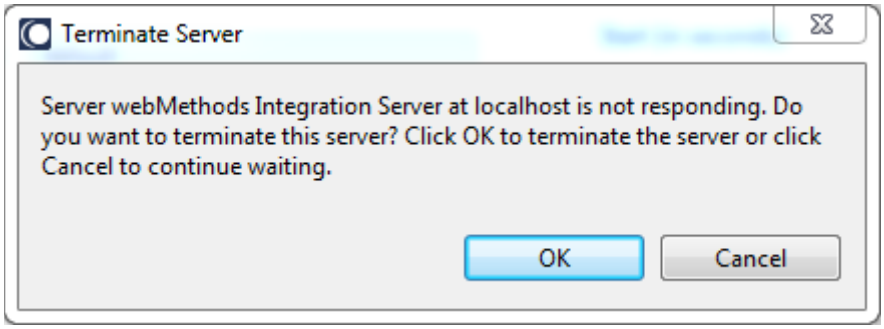
The start action executes a shell script designated to start the server. The executing script must block while the server remains started. The runtime environment for the server includes an environment variable to ensure the script blocks. Otherwise, if the server is started asynchronously, Designer will report an error immediately when the start executes. See the Server Fails to Start section for more details. Designer changes the state from "Stopped" to "Starting". Designer uses a polling mechanism to periodically ping the server. Once the server has been started and responds to the HTTP request, the server state transitions from "Starting" to "Started".

For the Integration Server, the IS server connection details defined in the Window/Preferences/Software AG/Integration Servers settings are used to connect to the Integration Server and execute a GET request using basic authentication. If the server returns the expected response code in the [200-300) range, Designer will change the state of the server from starting to a started state. This implies there should be an enabled server connection for the local profile with valid IS credentials.

Note: See the Server Fails to Start section for troubleshooting details.

3.4.3.2 Stop

The stop action executes a shell script for stopping the server. If the server fails to stop within the timeout period, the user may terminate the stopping server (or continue to wait) as shown in Designer.



Note: Termination does not affect the state of the actual server; it only resets Designer's view of it.

Designer's server shutdown detection uses a polling mechanism involving two steps. The first step concludes once the server fails to provide a valid response code to Designer's HTTP "ping" request. The server pinged periodically until it gets the expected failure or the stop timeout has been exceeded. The second step concludes once the OS process executing the shutdown script to terminates. This gives time for the server to achieve a complete shutdown and cleanup. Once the second step completes, the server state changes from "Stopping" to "*Stopped*".

3.4.3.3 Debug

When the server is started from Designer, the configured debug port for the server is always opened even when the server is not started with the "Debugging" action. In other words, there is no difference between how the server is started for the "*Started*" versus the "*Debugging*" action. The distinction between these two actions occurs when Designer is updating its server state. For the "*Debugging*" action, Designer will open a socket connection to the JPDA port to make debugging source code possible.

Note: If the server is launched outside of Designer, make certain it is launched in Debug mode; otherwise, it is not be possible for Designer to connect to the server in Debug mode if that JPDA port is not open. In this case, Designer will change the server state to "*Started*".

Note: If you are uncertain of the server's state, use an OS utility to see if the JPDA port is in LISTEN mode.

3.4.3.4 Restart

The "Restart" and "Restart in Debug" actions are a combination of the stop action followed by the "Start" or "Debug" action.

3.5 Project Publisher

This section describes all the steps that occur when an Application Platform project is built in Designer and published into the server. There are three major phases to publishing.

- Building the project
- Assembling the project into a module
- Deploying the module into the server

3.5.1 Project Builds

Before a project can be published it must first be validated and compiled. During the build phase, source code is compiled, additional metadata files may be produced, etc., but no additional bundle-related activities occur at this time.

3.5.1.1 Build Actions

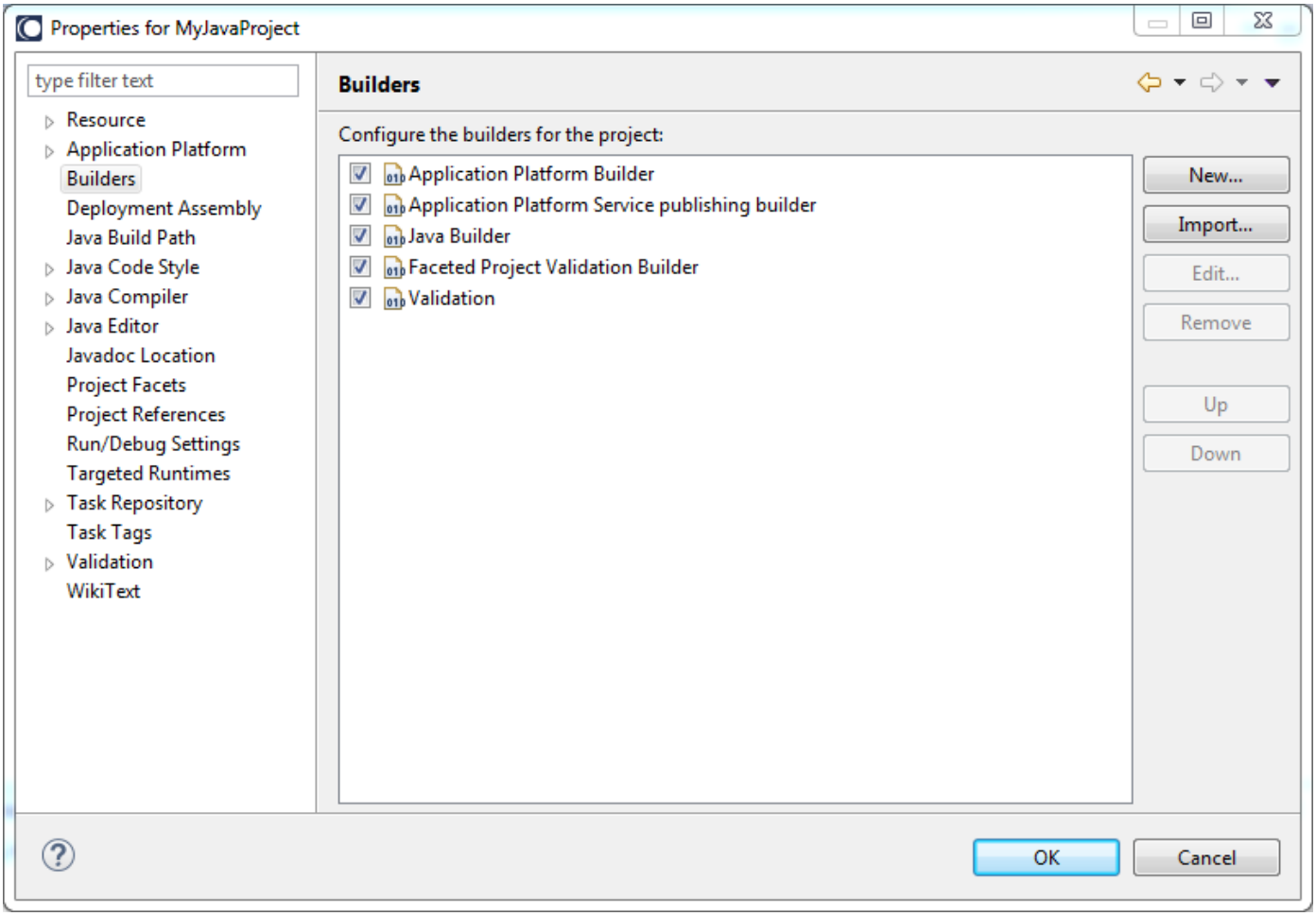
There are three Eclipse project build actions that are relevant to Application Platform project building.

- Clean – purges transient files for a project
- Incremental Build – execute build tasks for only those modified resources since a prior build
- Full Build – build or rebuild all project resources regardless of current build state

For more details, please consult the Workbench User Guide in the Designer Help Contents.

3.5.1.2 Application Platform Project Builders

Application Platform includes custom Eclipse project builders. These builders execute in the prescribed order as a means to silently perform Application Platform tasks on a project during the Eclipse build actions – clean, incremental build, and full build. This screenshot illustrates Application Platform's project builders.



3.5.1.2.1 Application Platform Builder

The Application Platform Builder is installed when the Application Platform Core facet is enabled for the project. It performs additional tasks in response to Eclipse build actions. For example, it passes additional context information captured during project compilation to the Project Publisher when it is time to assemble the project bundle.

Note: Do not disable or remove this builder from the project.

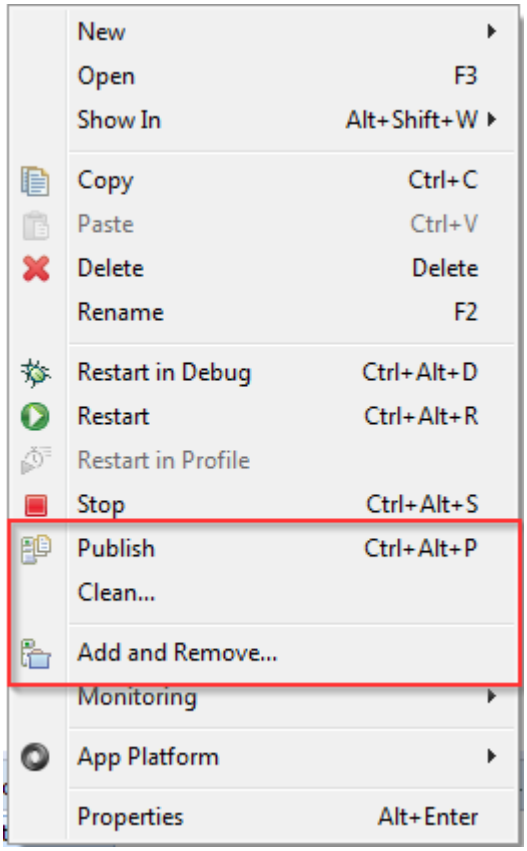
3.5.1.2.2 Application Platform Service Publishing Builder

This builder is installed when the Application Platform Core facet is enabled for the project. . It performs additional tasks such as creating additional files necessary to publish a project's services in the OSGi container.

Note: Do not disable or remove this builder from the project.

3.5.2 Server Operations for Projects

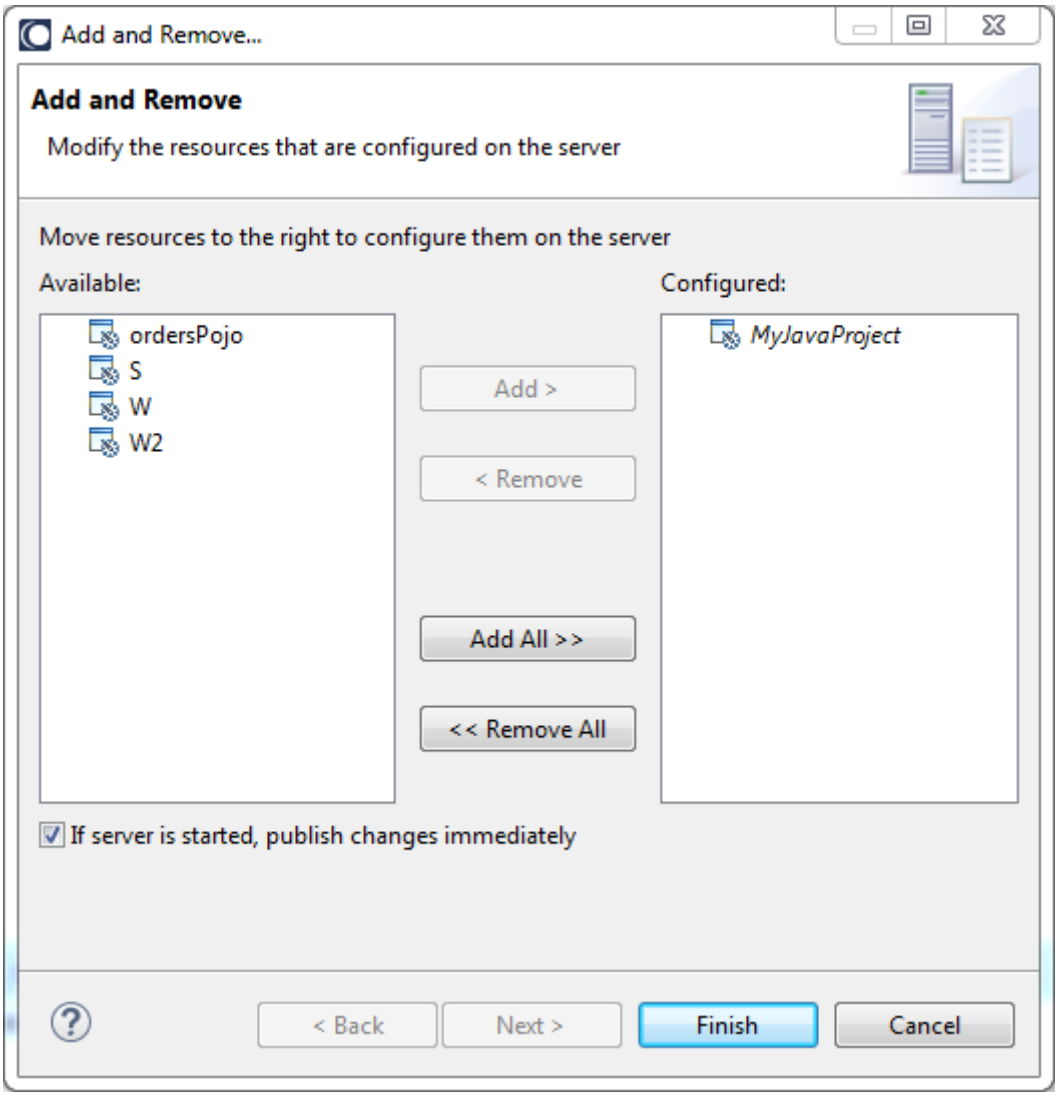
A project must be added to a server in the Servers view before it can be published into the server's OSGi container. The Servers view provides three actions to support this task.



3.5.2.1 Add or Remove Project

A project may be added to or removed from a server in the Servers view. Select the configured server and select "Add and Remove…" menu item from the Servers context menu shown above. This will present a new dialog where the user can add new projects or remove projects from the server.

In this example, *"MyJavaProject"* is selected from the left pane and moved to the right pane. Configured projects already published in the server, may be removed by moving them from the right pane to the left. The list of configured projects on the right will be published to the server when the Finish button is pressed.



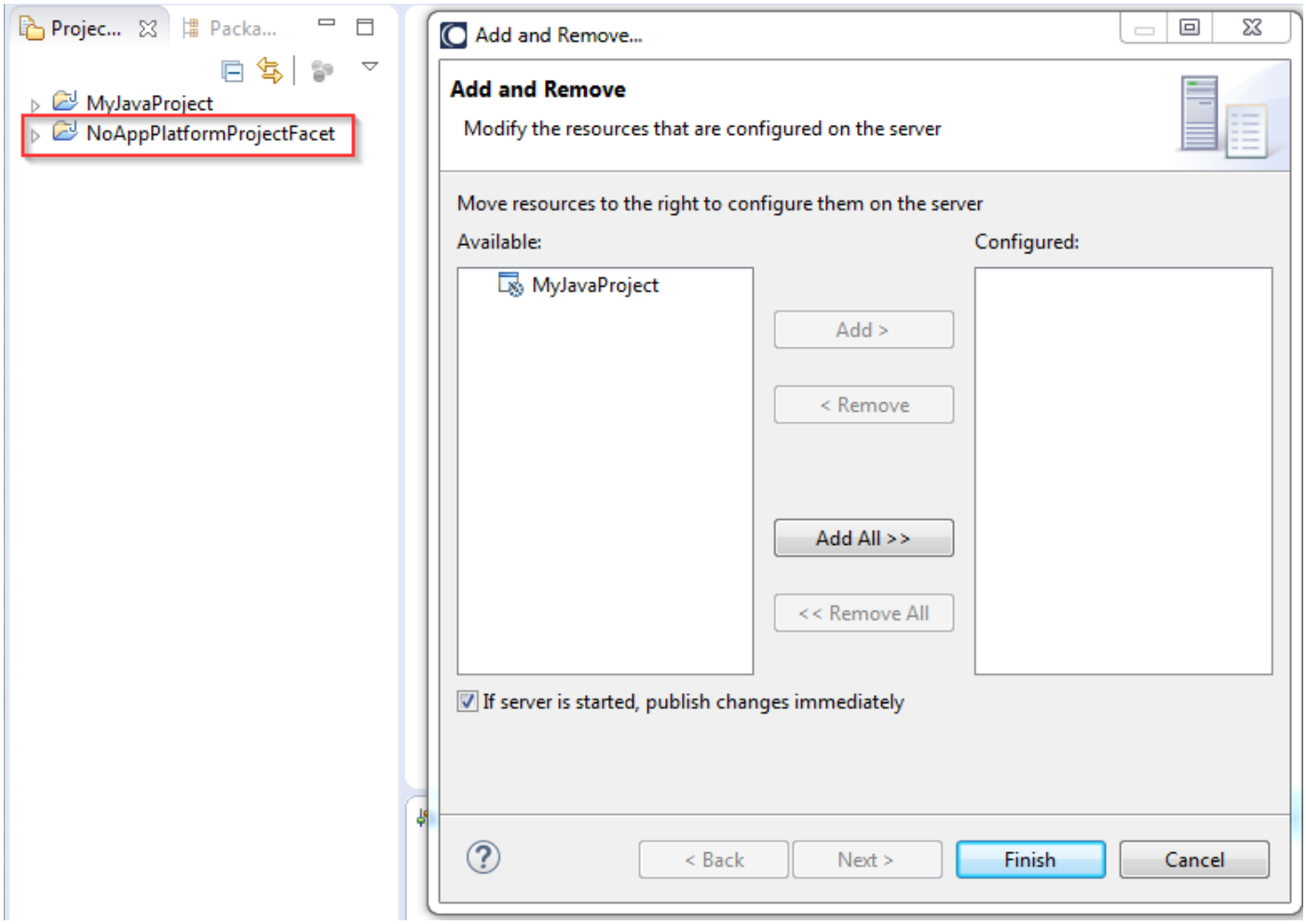
3.5.2.1.1 Publish Changed Immediately

If the "If server is started, publish changes immediately" check box is checked, Designer will immediately attempt to add or remove the projects as selected after clicking the Finish button. When the check box is unchecked, the user must manually execute a publish request to install the project bundle on the server.

Note: The checkbox also determines when projects are "unpublished". When this check box is unchecked and a project is removed from the server in the view shown above, the project's bundle is not removed from the server until the next time a publish operation is requested.

3.5.2.1.2 Available Projects

As demonstrated in this screen shot, not every project created in Designer can be published into the server. Notice there is a project called "*NoAppPlatformProjectFacet*" which is not included in the available list of projects for the *Add and Remove* dialog.



For a project to be eligible for installation into the server, the following must be true:

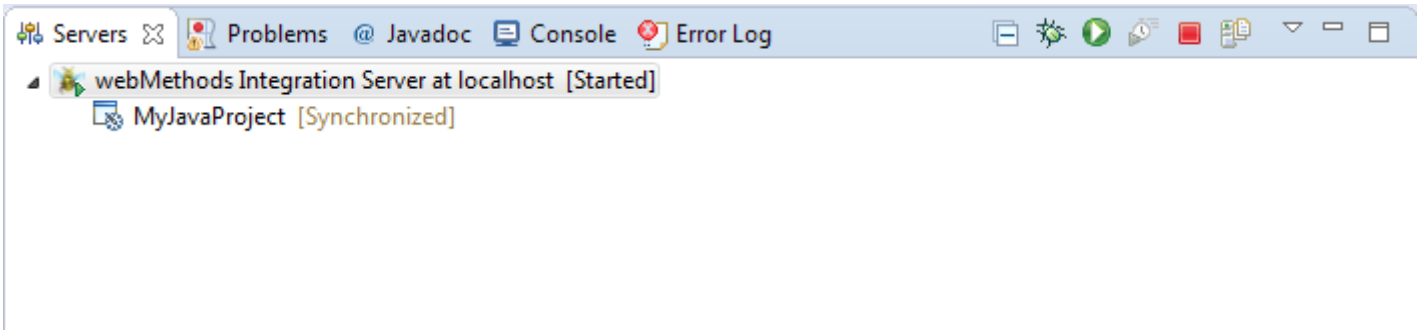
- Project must not be closed
- Project must have at least the Application Platform Core project facet

Once a project has been added to the server, it may still be removed from the server's *Configured* list even if the project is later closed in the explorer view.

Important: Do not delete a project from explorer that is published to the server. This will leave the project orphaned in the server with no easy means to remove it. See the section

called Manually Uninstall Bundle from Server for the manual steps required to remove a published project.

For example, this screenshot shows a server named "webMethods Integration Server at localhost" in a "*started*" state with a "*synchronized*" project named "MyJavaProject".



Note: For more details regarding server states and project statuses, please refer to the "*Server Views*" section of the "*Web Tools Platform User Guide*" under Designer's Help Contents.

3.5.2.2 Publish

The server publish operation will assemble a project bundle from the project files from the most recent build. Use this operation if making small incremental changes for the fastest results.

3.5.2.3 Clean

The clean operation will ensure the project is fully rebuilt and not incrementally published into the container.

3.5.3 Assemble the Project Bundle

Once the project has been built, it is time to assemble the project into a module – i.e. create an OSGi bundle. This process is divided into several steps as well.

- Create the OSGi manifest
- Create the jar and stage it in an Artifacts directory
- Copy the jar to the bundle repository

3.5.3.1 OSGi Manifest

The first stage of bundle creation pertains to the OSGi manifest for the project. If an OSGi MANIFEST.MF file is included in the project's *src/main/resources/* folder, it will serve as a template during bundle creation. Otherwise, a manifest is created automatically. Often the default manifest will be sufficient for many projects; however, at times it will be necessary to create a custom

manifest. A couple of examples where this will be necessary is included to convey the point, but this list is not complete.

3.5.3.1.1 Indirect Package Imports

The default OSGi manifest is created with a list of package imports by analyzing the project classes' imports to locate external package dependencies found in the bundle. Since everything the project is compiled against is also a bundle, the analysis can match package imports to specific bundle versions. Sometimes dependencies are declared in additional metadata such as XML files and invoked indirectly. (e.g. class references in web.xml descriptors.) These additional dependencies must exported by another bundle in the container, and a custom manifest must be produced so the additional package(s) may also be imported.

3.5.3.1.2 Reduced Scope of Package Exports

By default all packages defined in the project bundle are exported. If this is undesirable, then an alternate set of exports may be declared in a custom manifest. For example, it is considered an OSGi best practice to only export packages representing a public API while the implementation packages remain private to the bundle.

3.5.3.2 Artifacts directory

Once the project has been completely compiled, its contents are inserted into a bundle jar and copied to a working directory outside of the project's workspace. The location must be outside of the workspace because Eclipse locks the entire workspace while building projects. This special "*artifacts*" directory can be found in the following location: ${user workspace}/.metadata/.plugins/com.softwareag.ide.eclipse.pld.bundle.builder.ui/${project name}/artifacts/.

When executing the Create Project Bundle context menu, the project bundle is copied to this location. The bundle will be deleted from the artifacts directory after a successful clean build action.

Note: The location of the artifacts directory is an implementation detail documented here for diagnostic purposes. Do not depend upon its existence or location as these details could change in the future.

3.5.3.3 Repository directory

When publishing a project to a server, the project bundle is copied from its artifacts directory to the repository directory in the server. This directory is the active location for all customer-developed projects published to the server. The repository directory can be found in the following location: ${Application Platform install home}/profiles/IS_default/workspace/app-platform/deployer/bundles/. This relative path is not configurable. When project bundles are published or unpublished from the container, they are either added or removed from the repository directory.

Note: "*IS_default*" is the only instance name supported at this time.

3.5.3.4 Deployment

The server uses an OSGi service provided by the Common Platform which is used to install and/or remove bundles in the repository directory. Any errors are returned by this server-side service to the Project Publisher and updated in the Designer's error view.

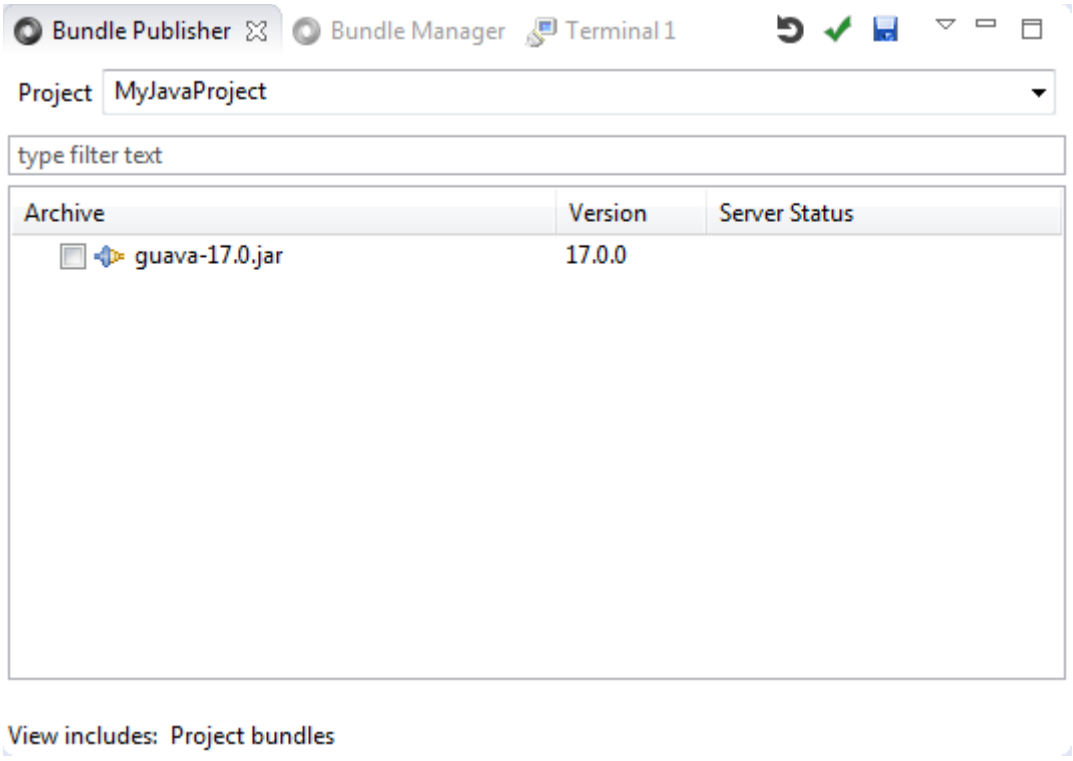
Note: The server does not support dynamic "hot" deployment. One cannot install or remove bundles by simply adding or removing files in this directory.

3.6 Managing Dependencies

Applications are usually composed of more than just one or more source projects developed in Designer. Projects may have dependencies to 3rd party bundles not currently in the platform as well as bundles produced by other teams. Two views in the Application Platform facilitate this purpose. *The Bundle Publisher* view is used to publish (or unpublish) a bundle into (or from) a container. The *Bundle Manager* view is used to create a bundle from a plain jar – i.e. a jar that is not an OSGi bundle.

3.6.1 Bundle Publisher

The *Bundle Publisher* provides a convenient mechanism for installing or removing additional bundles into the server from Designer.



Note: Bundle Publisher is not used to publish a project's bundle into the server. The Servers view should be used for that activity.

Important: Consult Software AG documentation for details regarding the *Asset Builder Environment* and *Deployer* components. These tools should be used to deploy assets outside of Designer – e.g. automated builds.

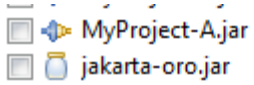
3.6.1.1 View Contents

The list of items displayed in the view comes from the following locations

- A selected project's classpath
- A directory configured for the Bundle Manager preferences settings.
- The server runtime container

Depending upon the Bundle Publisher's preference settings, certain categories of items may be excluded from the display. See the Bundle Publisher section for more details.

The view uses different icons to distinguish between plain Java jars and OSGi bundles. In the example below, My-Project-A.jar is an OSGi bundle whereas Jakarta-oro.jar is a plain Java jar.

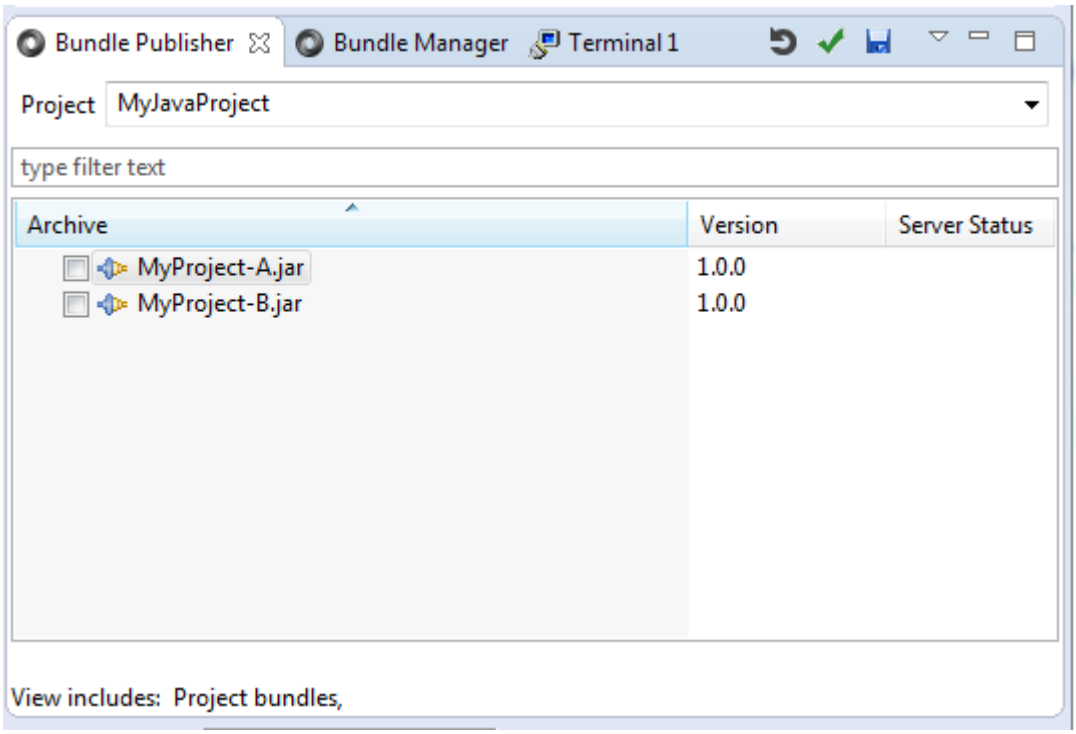


3.6.1.2 Server State and View Checkboxes

The Bundle Publisher uses the checkbox state to perform publish and unpublish operations from one view at the same time. This is best described with an example. The next sections will demonstrate the various combinations in which bundles may be installed or removed.

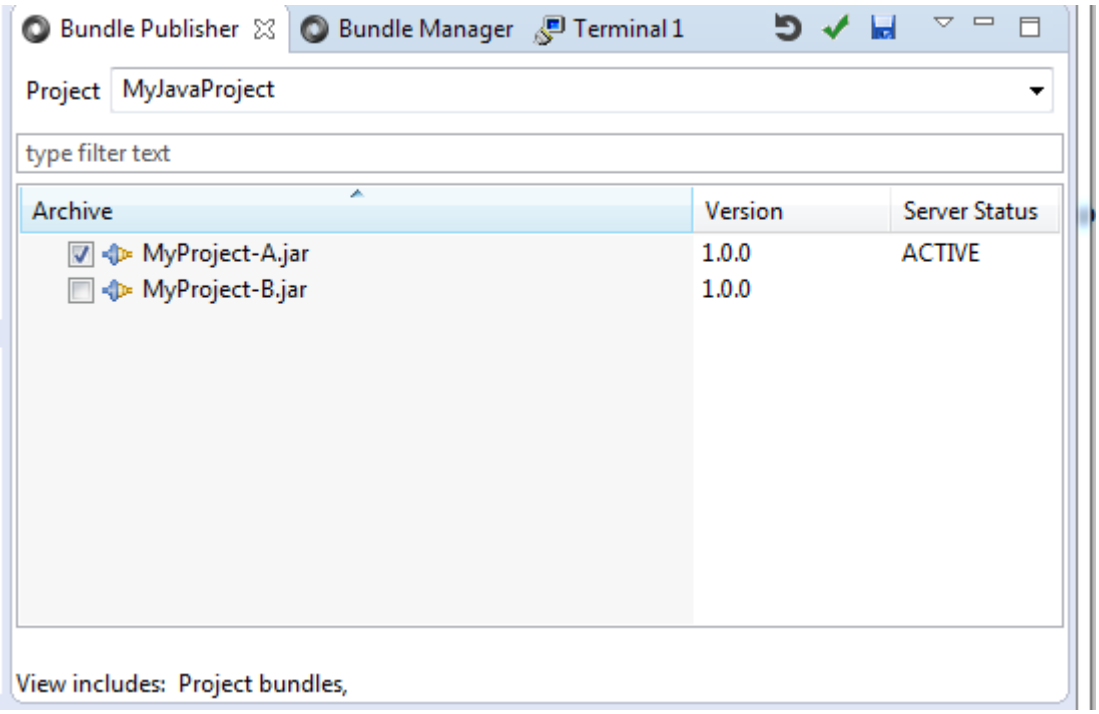
3.6.1.2.1 Initial View

Assume there are two bundles residing in a directory included in the Bundle Manager's configuration. Neither bundle has been installed in the server. There is no server status, and neither bundle is checked.



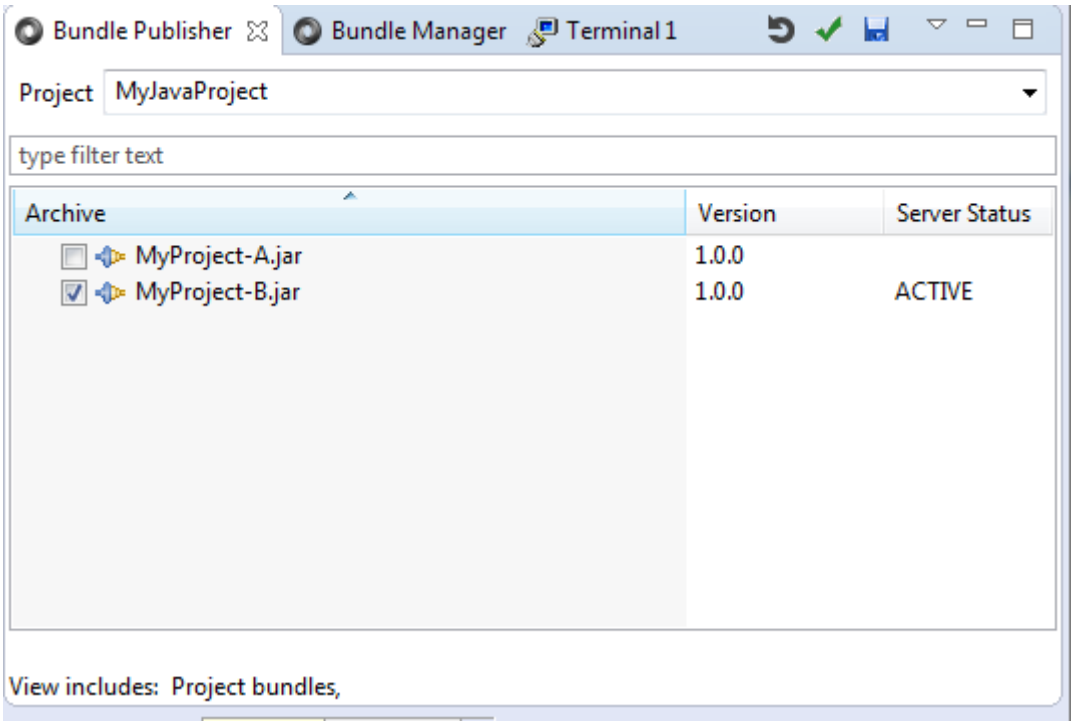
3.6.1.2.2 Publish A Bundle

The user selects a bundle by checking it, and presses the server update toolbar action. Now the view shows that MyProject-A has a server status. Notice the bundle is still checked after the view has refreshed. The checked state implies this bundle is already published in the server.



3.6.1.2.3 Bundle Publish and Unpublish Another Bundle

Next, the user decides to unpublish the MyProject-A bundle by *unchecking* the published MyProject-A bundle, and at the same time checking the MyProject-B bundle. So to be clear, two actions are performed before the user clicks the server update toolbar action. After pressing the update button (i.e. the floppy disk icon), the view refreshes to the following state. Notice MyProject-A bundle is no longer checked whle MyProject-B is checked.



Using the checkbox idiom to reflect the current deployment status of a bundle provides the means to quickly alter a collection of bundles. This is very useful when generating a dependency graph to include or remove bundles from a collection.

3.6.1.3 Dependency Graphs

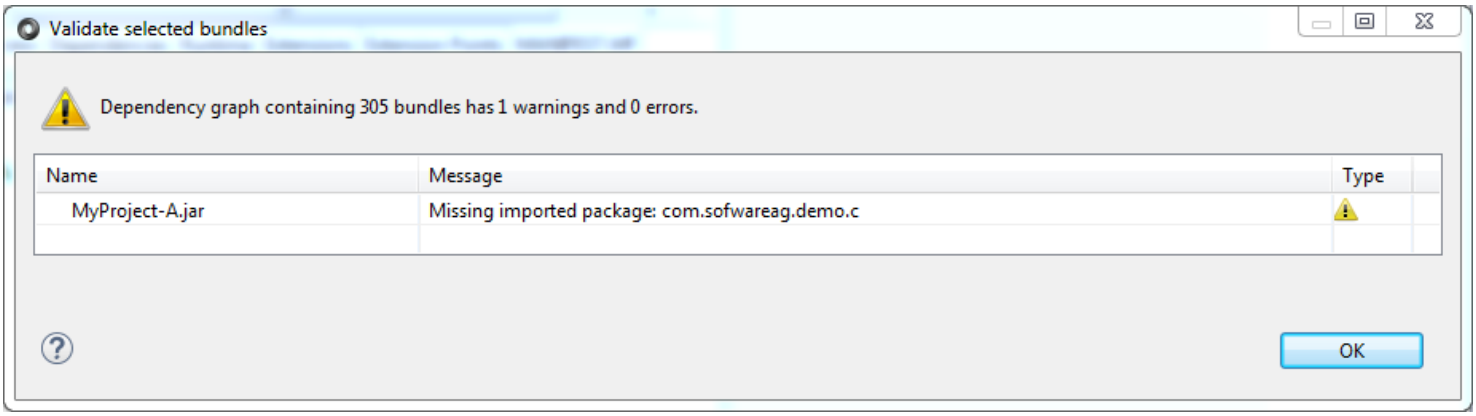
The Bundle Publisher uses a dialog to convey potential warnings and errors that are discovered when validating a collection of bundles. The group of bundles is used to form a *dependency graph* by examining all of the OSGi manifests in the collection to determine dependencies between the bundles. When the server is started, those active bundles are also included in the graph unless they are explicitly removed. Several examples are provided to explain the purpose for this validation.

3.6.1.3.1 Publish a Bundle with Missing Dependency

Assume there are two bundles named MyProject-A and MyProject-B, and neither have been published to the server. MyProject-B has a dependency upon MyProject-A and MyProject-A has a dependency on a package (com.softwareag.demo.c) that is not exported by the project or any bundle in the server. This represents a missing dependency. Attempts to publish MyProject-A will fail since the import is required.

	Imports Package	Exports Package
MyProject-A	com.softwareag.demo.c	com.softwareag.demo.a
MyProject-B	com.softwareag.demo.a	com.softwareag.demo.b

In this scenario, selecting the two bundles above and performing a validation will return a warning message as follows.

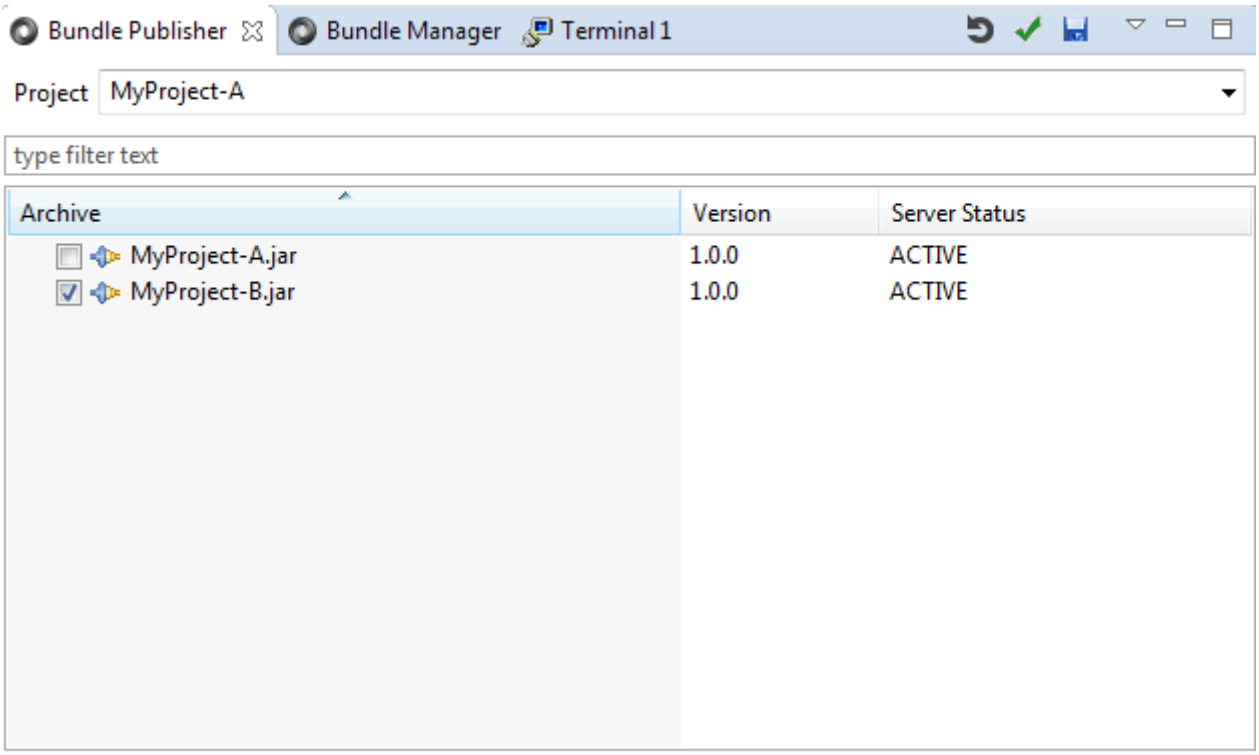


3.6.1.3.2 Remove a Bundle that Provides a Dependency

Assume there are two bundles named MyProject-A and MyProject-B. MyProject-B has a dependency upon MyProject-A and both have been published to the server.

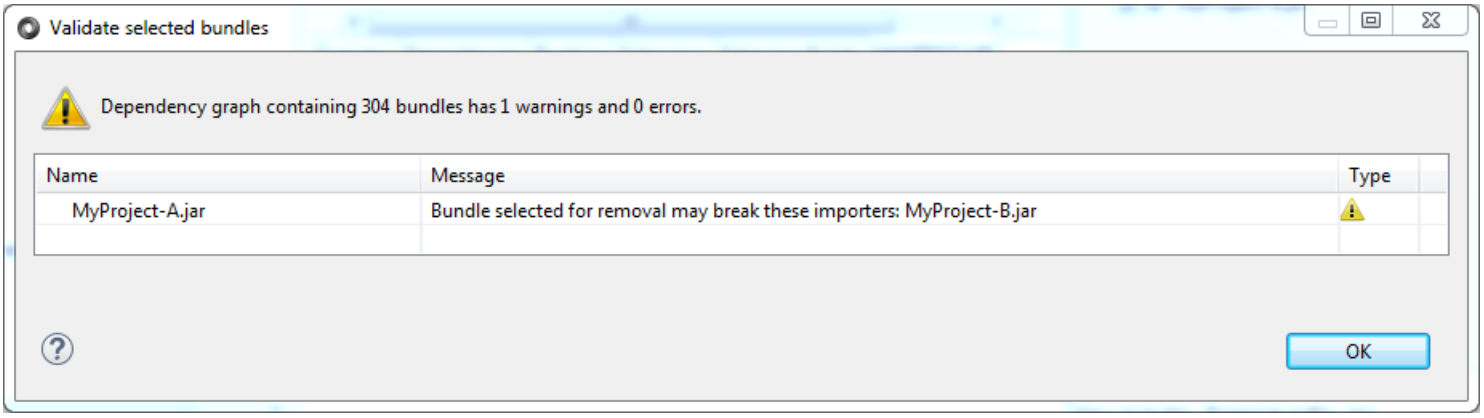
	Imports Package	Exports Package
MyProject-A		com.softwareag.demo.a
MyProject-B	com.softwareag.demo.a	com.softwareag.demo.b

Unselecting MyProject-A (by unchecking its checkbox)



View includes: Project bundles, Jars

Pressing the validation toolbar action will produce a warning message because MyProject-B is still installed on the server and removing MyProject-A will result in a missing dependency.

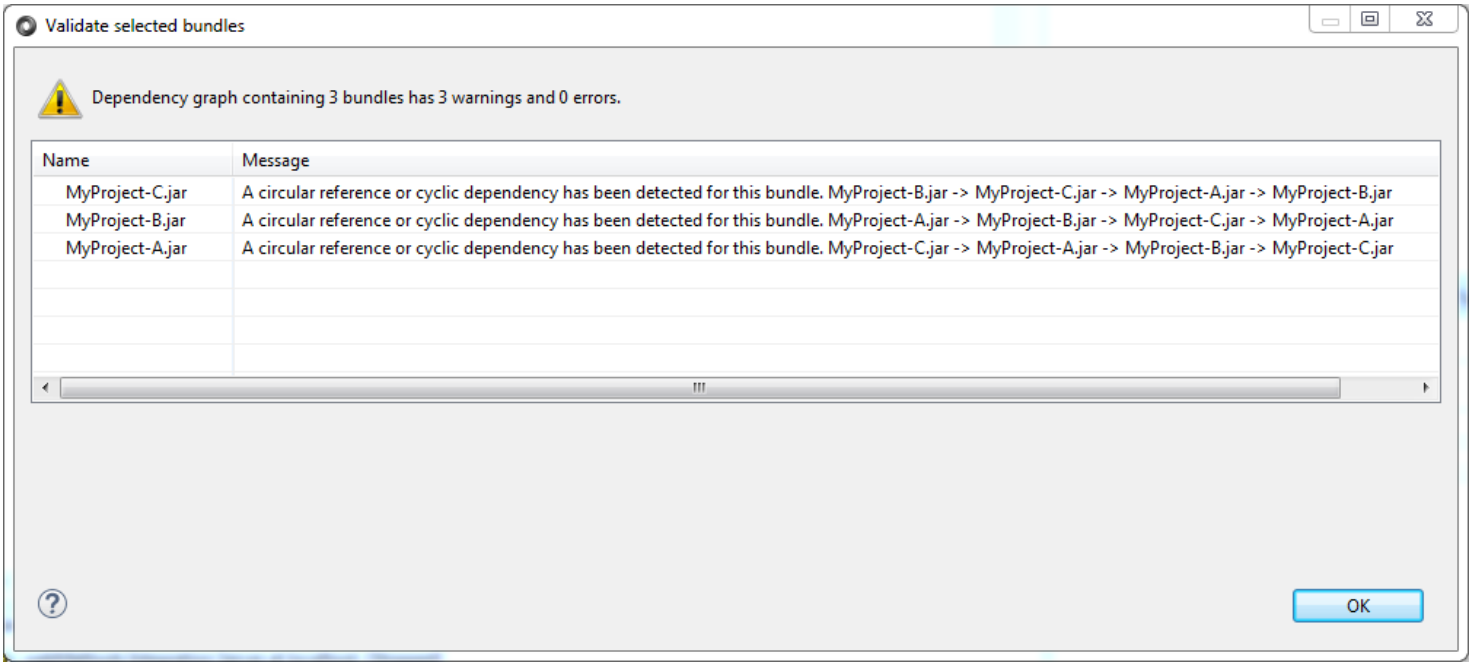


3.6.1.3.3 Circular Dependencies

Assume there are three bundles named MyProject-A, MyProject-B and MyProject-C, and none have been published to the server. MyProject-A has a dependency upon MyProject-B and MyProject-B has a dependency on MyProject-C. MyProject-C has a dependency upon MyProject-A which represents a circular dependency. Attempts to publish these bundles will fail.

	Imports Package	Exports Package
MyProject-A	com.softwareag.demo.b	com.softwareag.demo.a
MyProject-B	com.softwareag.demo.c	com.softwareag.demo.b
MyProject-C	com.softwareag.demo.a	com.softwareag.demo.c

In this scenario, selecting the two bundles above and performing a validation will return a warning message as follows. So in the first warning message, MyProject-C bundle is a dependency for Myproject-B. MyProject-C depends upon MyProiect-A which depends upon MyProject-B which completes the circular reference.



3.6.1.3.4 Message Details

There are three kinds of messages displayed in the dialog.

- Info Messages – Status messages conveying information
- Warning Messages – Usually implies a dependency issue that may prevent bundles from reaching an active state
- Error Messages – This implies the bundle is invalid. E.g. a corrupt file or jar with an invalid OSGi manifest

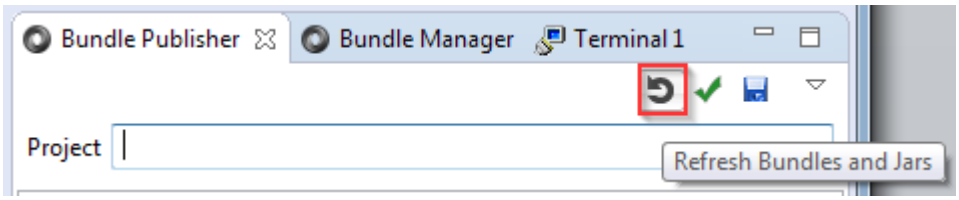
3.6.1.3.5 Dependency Checks

The following validation checks are performed when creating a dependency graph for a collection of bundles.

Warning	Notes
Attempt to publish a bundle that exports the same package and version.	
Attempt publish a bundle that imports a package that is not exported.	Note, there is a Bundle Publisher preference setting to ignore optional missing imports. i.e. Bundles with package imports containing this manifest qualifier: ;resolution:=optional configuration
Attempt to unpublish a bundle that exports a package imported by another bundle.	
Attempt to unpublish a bundle when one more bundles require it.	
Attempt to publish a bundle that will produce a cyclic dependency or circular reference.	

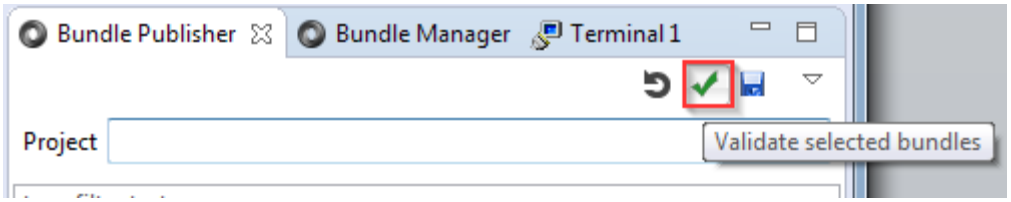
3.6.1.4 Refresh Publisher View

The Refresh toolbar action is used to refresh the contents displayed in the Bundle Manager view.



3.6.1.5 Validate Bundles

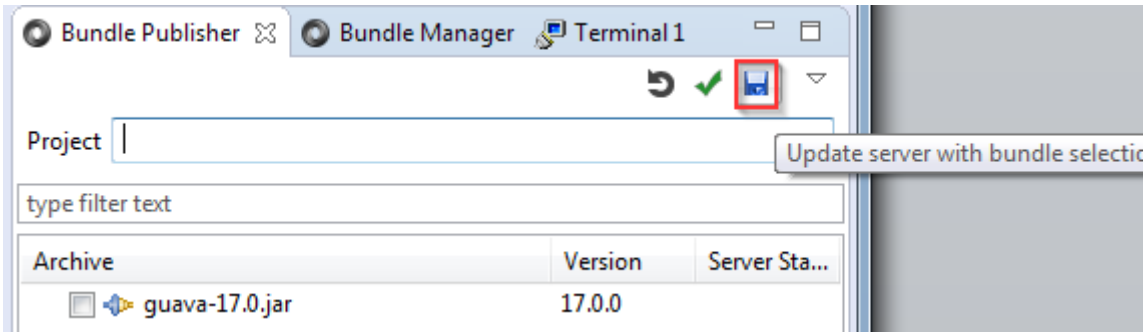
The Validate Bundles toolbar action is used to perform a dependency check on a selection of bundles. Its purpose is to catch potential errors before an attempt to publish to the server is performed. A dependency graph is produced for the checked bundles based upon each bundle's declared package import and exports. When the server is started, the collection of checked bundles will include those items which are currently published to the server. This gives the opportunity to test the potential impact to the server when bundles are removed.



Note: The text filter must be cleared before validation or update toolbar actions may be invoked.

3.6.1.6 Update Server

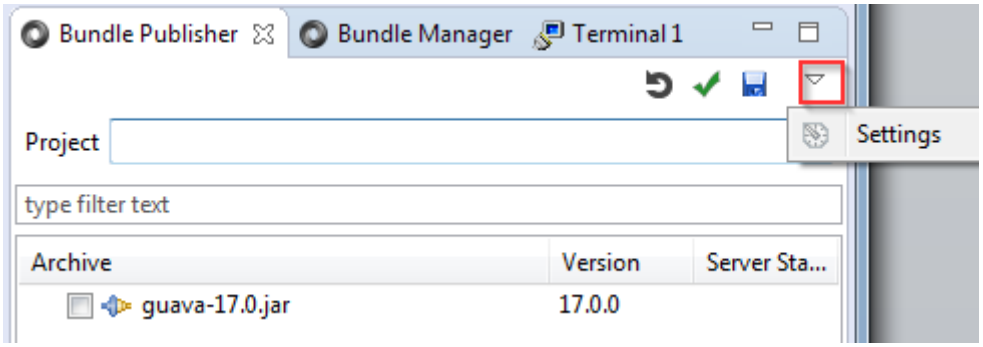
The update server toolbar action is used to publish or unpublish selected bundles into the server.



Note: The text filter must be cleared before validation or update toolbar actions may be invoked.

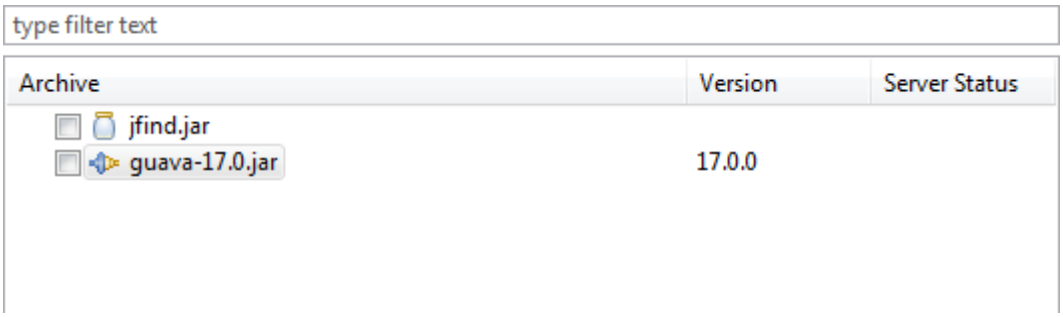
3.6.1.7 Configure Bundle Publish Settings

This configuration settings toolbar action launched the preference settings dialog for the Bundle Publisher. Please refer to Bundle Publisher section for configuration details.



3.6.2 Bundle Manager

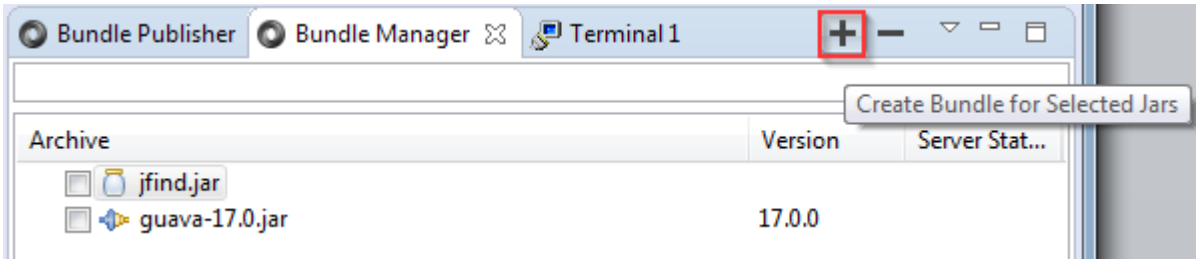
The Bundle Manager view is used to create bundles from non-OSGi jars. The screenshot below shows a "jfind.jar" plain jar and a "guava-17.0.jar" bundle and are used to illustrate the functions described in the next sections.



3.6.2.1 Create Wrapper Bundle

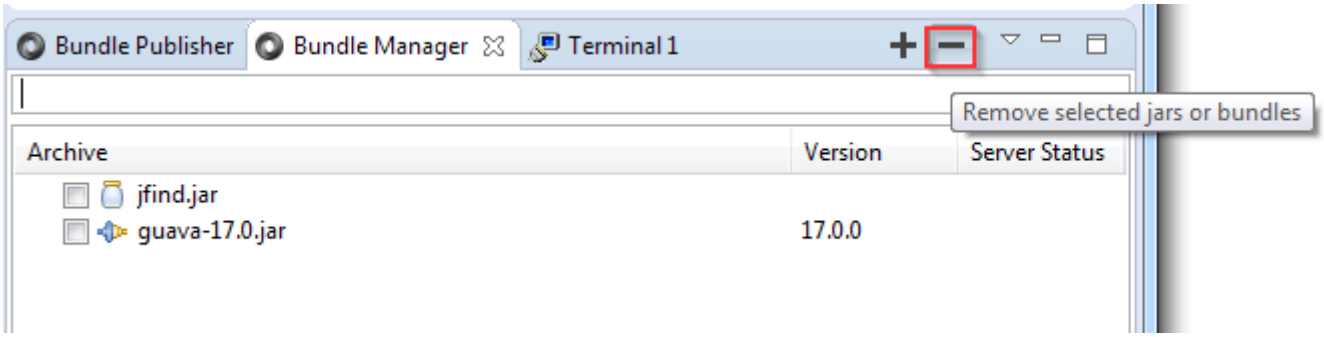
This toolbar action is used to create an OSGi bundle to host an ordinary jar so it may be published into the server. In the screenshot below, an OSGi bundle (guava-17.0.jar) and a plain jar (jfind.jar) are shown. Notice the icons are different for each type. Attempts to create a wrapper bundle while selecting the guava bundle will result in an error dialog.

One or more plain jars may be selected when invoking this action. In that case, all of the selected jars will be included in the wrapper bundle.



3.6.2.2 Delete Bundle

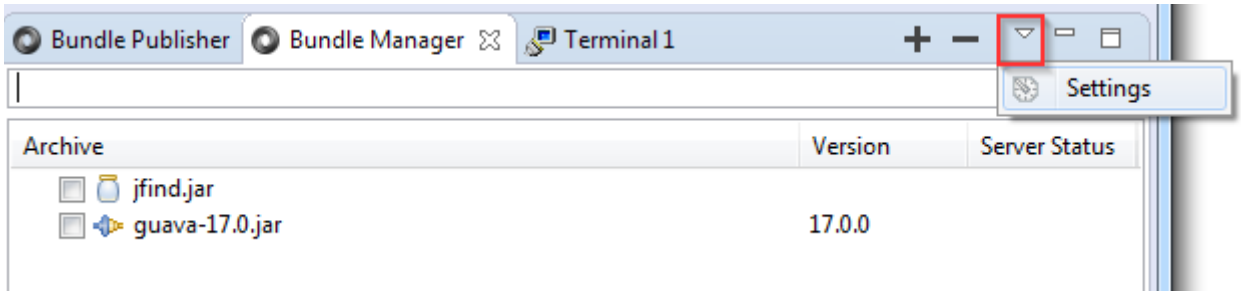
This toolbar action is used to delete a bundle or jar from the any of the directories defined for the Bundle Manager's configuration settings. Bundles that are published into the development server must first be unpublished via the Bundle Publisher before they are deleted in this view.



Note: The server must be started before bundles may be removed.

3.6.2.3 Configure Bundle Manager Settings

This configuration settings action configures the preference settings for the Bundle Manager. Please refer to the Bundle Manager section for configuration details.



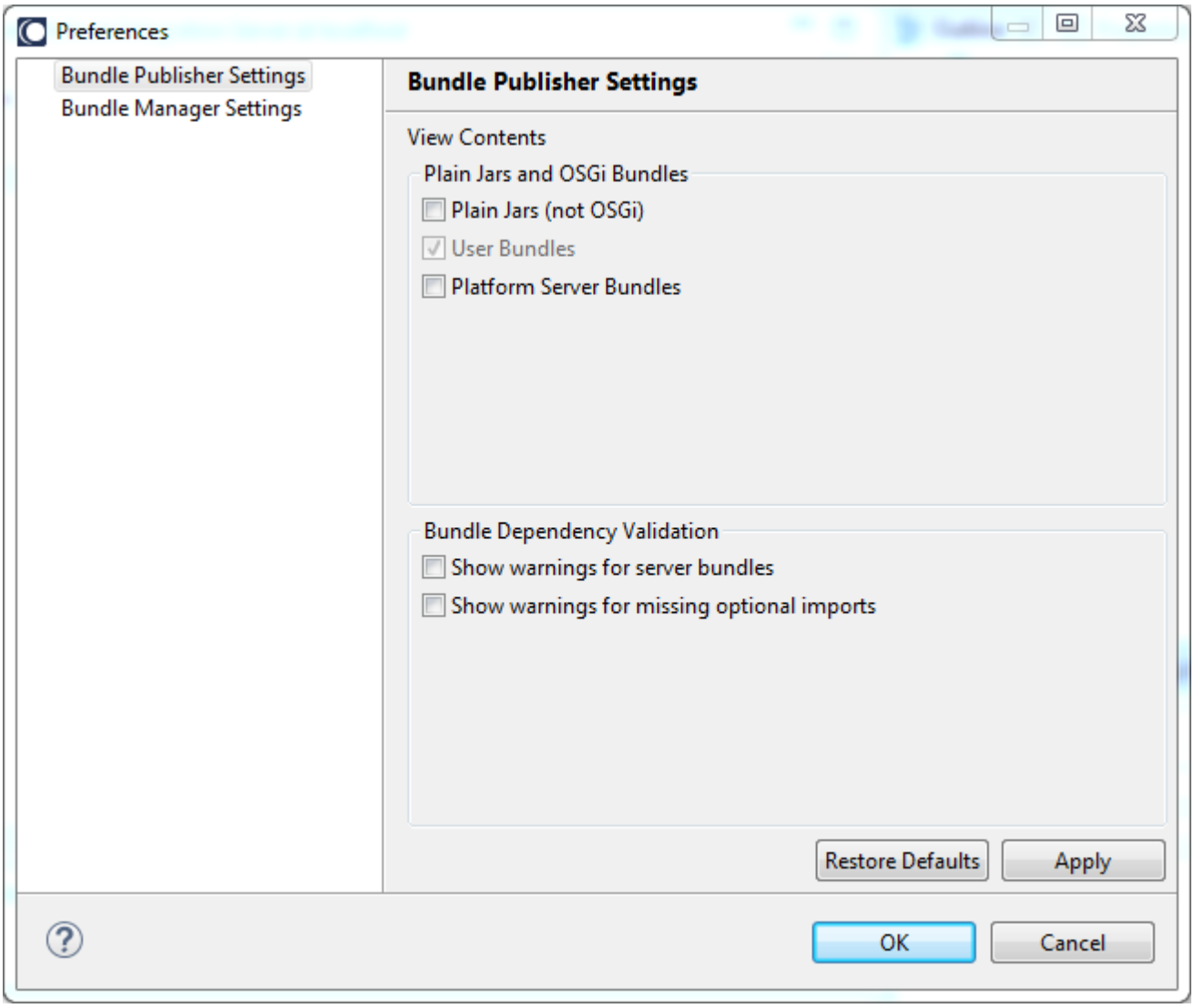
3.7 App Platform Configuration

Application Platform configuration is supported for the following elements

- Bundle Publisher View
- Bundle Manager View
- Eclipse Capabilities
- Servers View
- Project Configuration
- Customer Applications

3.7.1 Bundle Publisher

The Bundle Publisher configuration is divided into two sections. View contents determine what items are shown in the Bundle Publisher view. The *View Contents* section contains check boxes to provide fine-grained control over the contents included in the view. The *Bundle Dependency Validation* section may be used to limit the amount of content that is returned to the user during bundle validation or conveying bundle changes to the server.



3.7.1.1 Plain Jars

When the "*Plain Jars (not OSGi)*" checkbox is enabled, plain jars (i.e. jars which are not OSGi bundles) will be included in the Bundle Publisher view.

Note: Plain jars are not eligible for publish, so all plain jars are not eligible for selection in the view.

3.7.1.2 User Bundles

The "*User Bundles*" checkbox is included simply for the sake of clarity. By default, the view will always display bundles included in the selected project as well as any bundles that reside in one of the directories configured for the Bundle Manager. Therefore it is always enabled and cannot be unchecked.

3.7.1.3 Platform Bundles

Check the "*Platform Server Bundles*" checkbox, to see a list of the platform bundles which are delivered with the server profile. These bundles should not be removed; therefore, the Bundle Publisher view excludes these bundles by default. When this checkbox is enabled, the additional server bundles will be included in the view.

Note: Enabling this checkbox will add dozens of platform bundles to the Bundle Manager view.

3.7.1.4 Server Bundle Warnings

The Bundle Publisher only displays OSGi manifest warnings for active server bundles when the "*Server Bundle Warnings*" is enabled.

Note: Enabling this checkbox may produce excessive warning messages in unrelated bundles when publishing or validating bundles.

3.7.1.5 Optional Imports

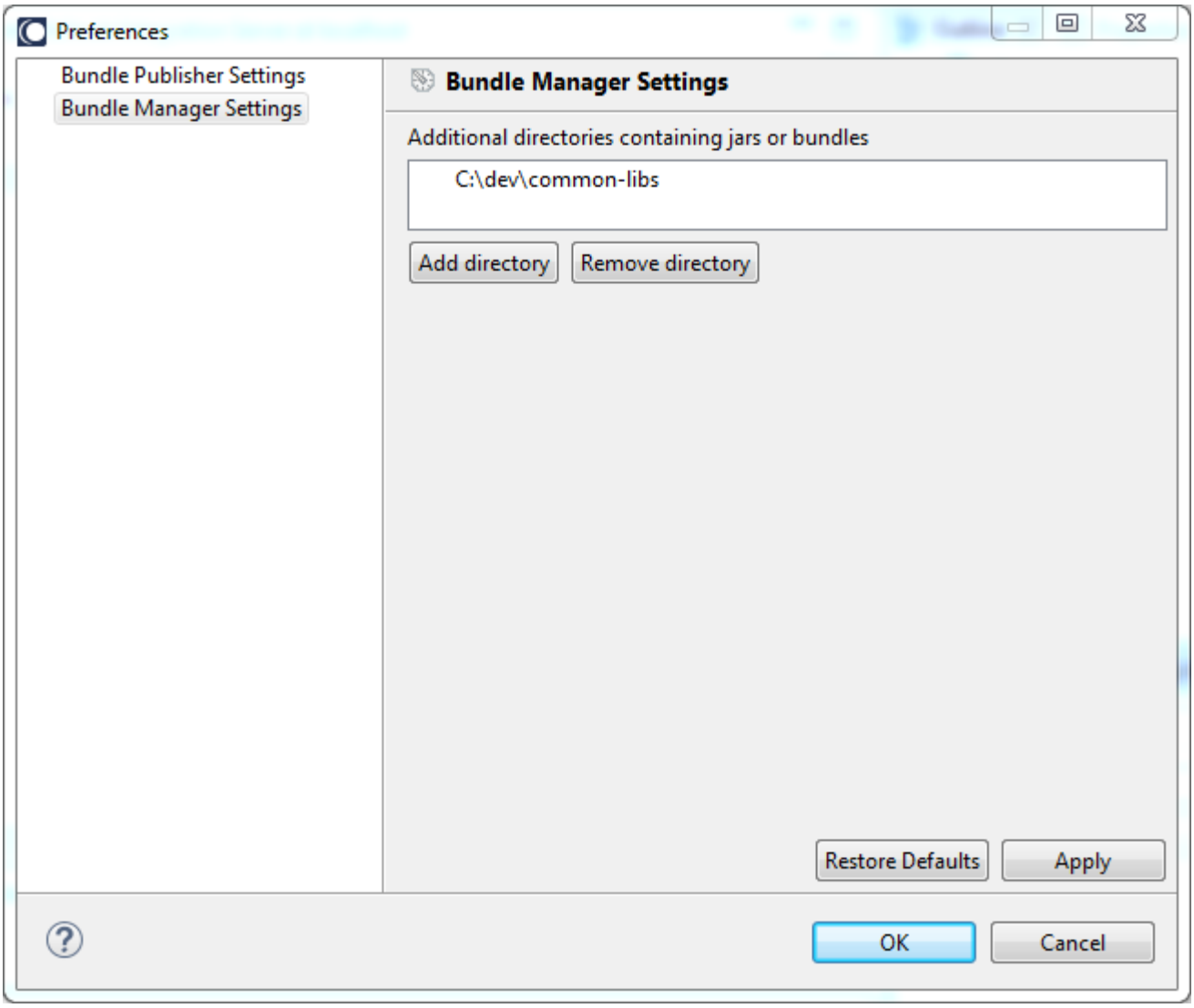
Check the "*Show warnings for missing optional imports*" checkbox to include warning messages when imported packages for a bundle are not included. These warnings are displayed when Bundle Publisher is validating bundles or applying updates to the server. If one of the bundles in the set has a MANIFEST.MF containing an Import-Package header with the "*resolution := optional*" qualifier, and no active bundle exports this package, a warning will only be returned if this check box is enabled.

Note: There may be a valid case where a missing package does not cause trouble – e.g. imports to test frameworks.

3.7.2 Bundle Manager

The Bundle Manager configuration settings is used to define one or more directories that may contain additional bundles to install into the server. Bundles that reside in one of these directories may be shared across user projects.

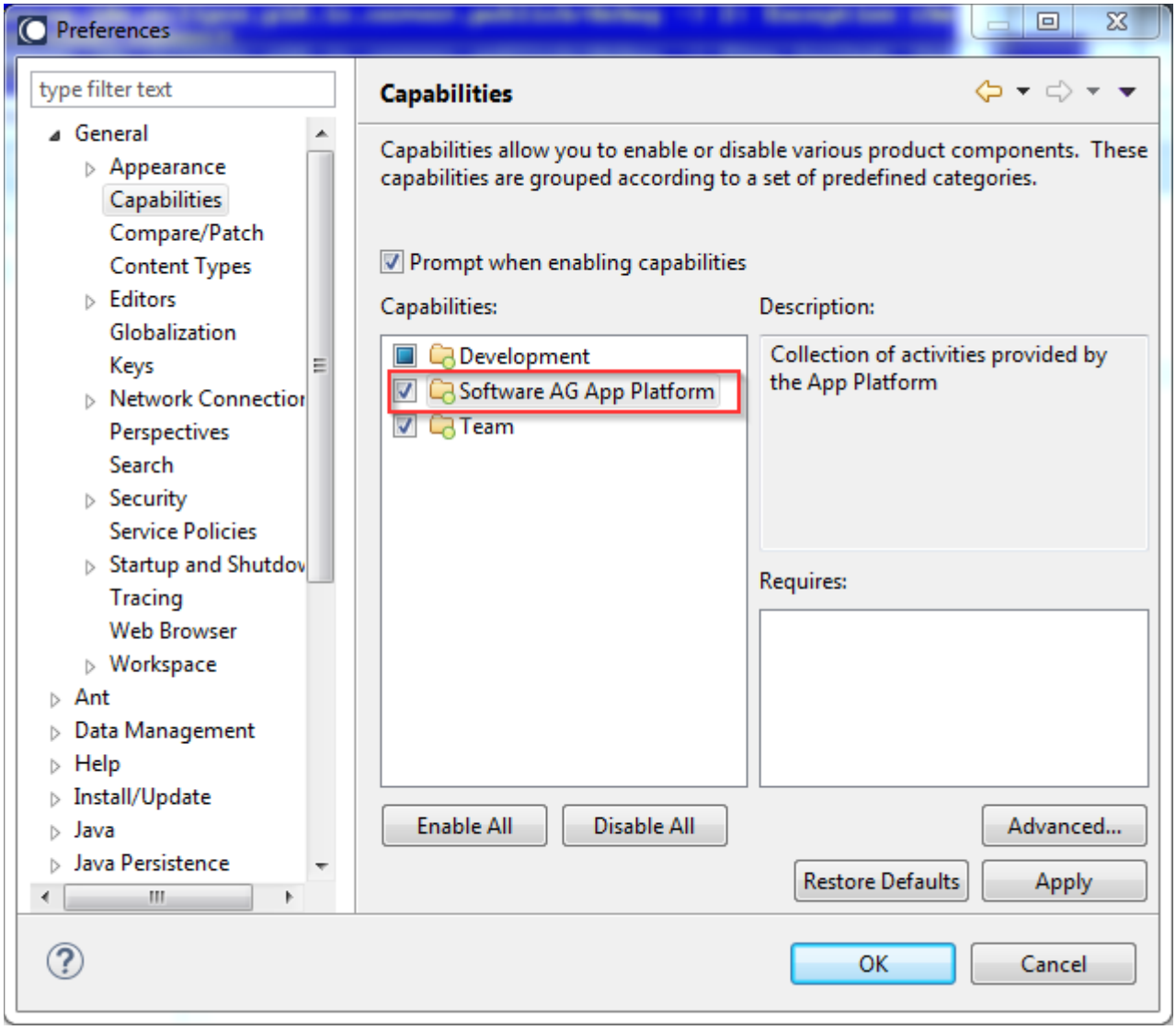
The screenshot below shows one directory added to the preference settings.



3.7.3 Capabilities

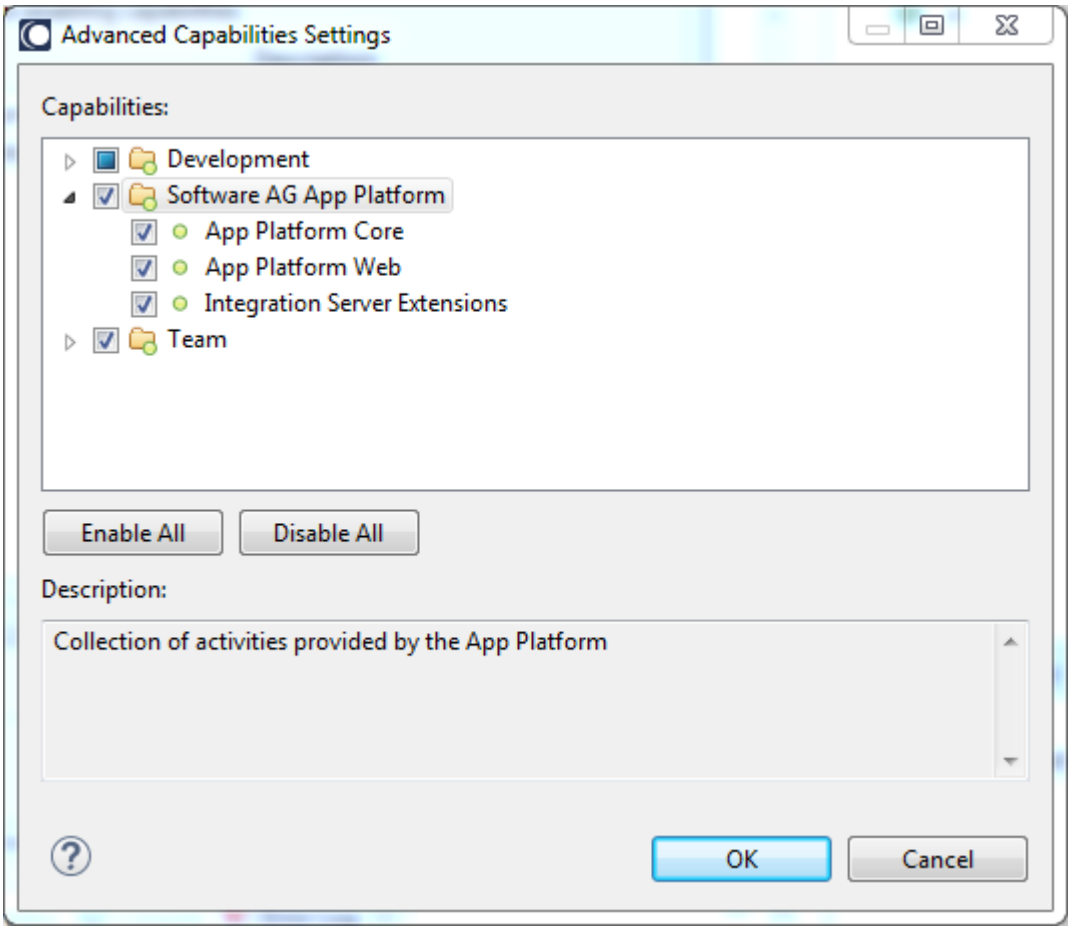
Eclipse capabilities are used to associate a collection of views or activities to a specific purpose. For example, Eclipse includes a J2EE capability including many wizards and views which are useful when working on J2EE projects. Once defined, a capability can be used to quickly hide those related items.

The Application Platform defines its own capabilities as well. The configuration settings are found under the Window/Preferences/General/Capabilities menu path.



Note: Unchecking the *App Platform* capability will cause the *App Platform* perspective to become hidden.

Press the "*Advanced…*" button to display the next configuration view. For example, uncheck the Integration Server Extensions checkbox to hide Application Platform features such as the IS Service Wizard.

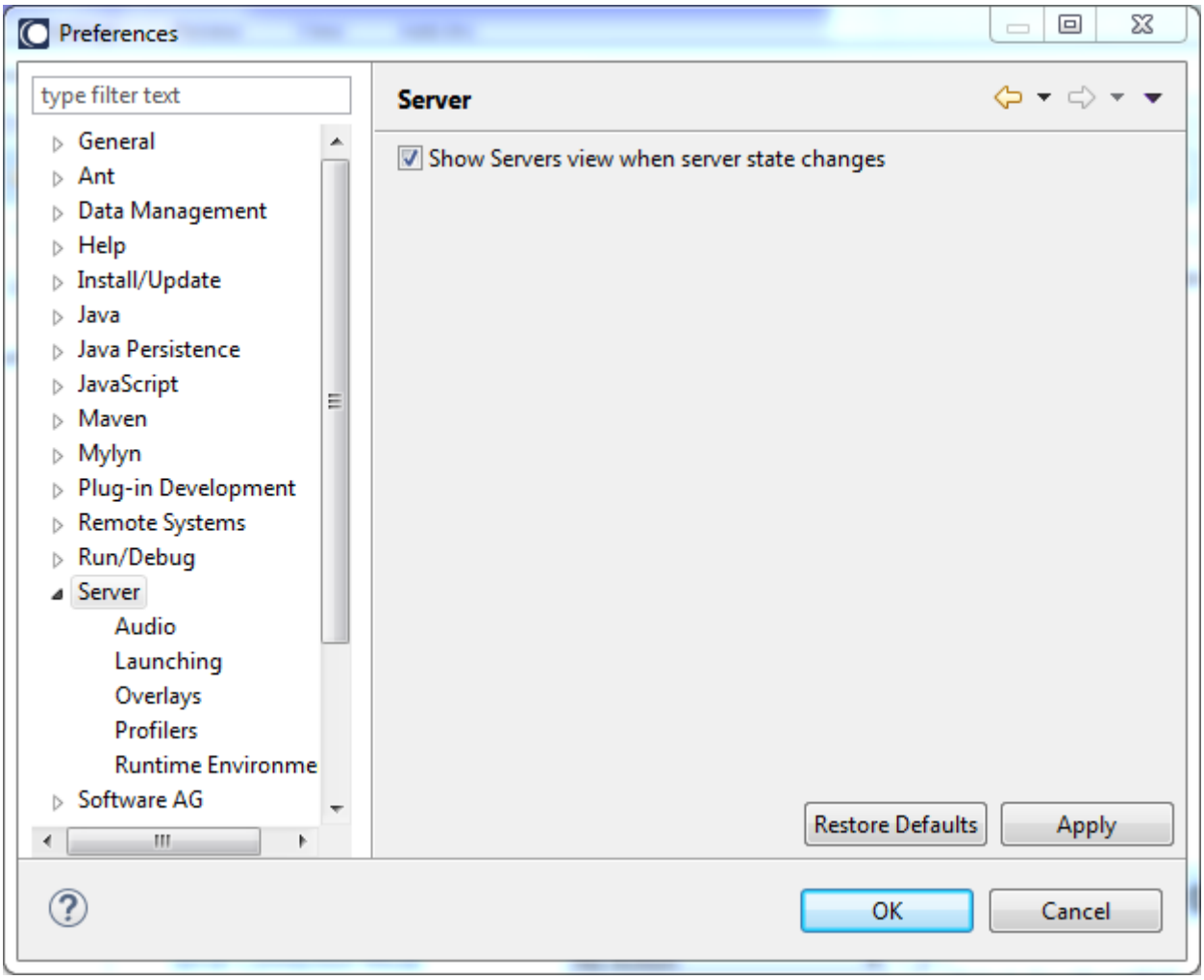


3.7.4 Server View Configuration

There Servers view has configuration preferences that are accessible under Window/Preferences/Server menu.

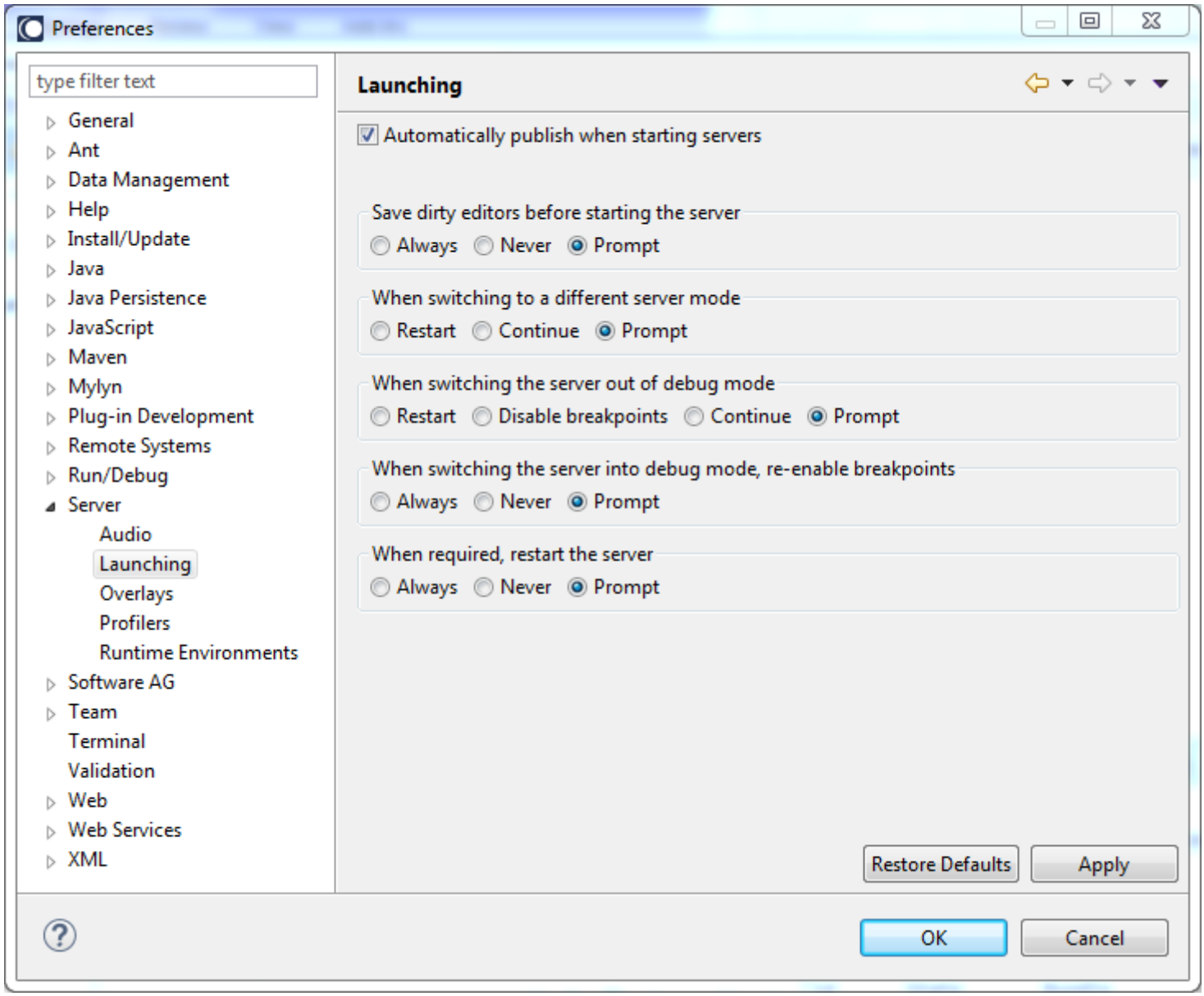
3.7.4.1 Servers View Updates

When this check box is enabled, focus will be redirected to the Servers whenever there is activity during startup, shutdown or project changes in the server.



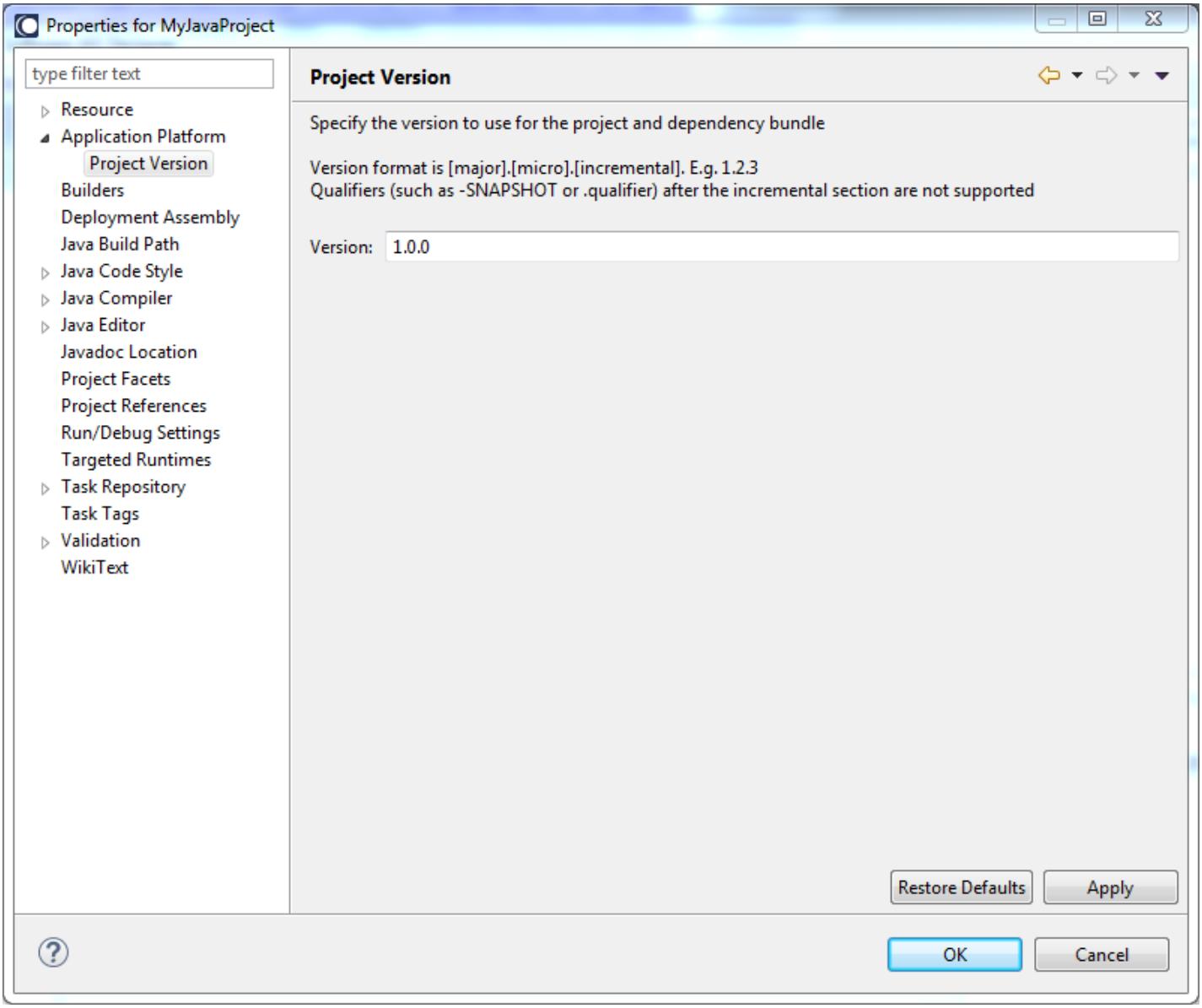
3.7.4.2 Server Launch Preferences

The Eclipse WST project offers configuration for the items shown below. For more details regarding these preference settings, please consult the *Web Tools Platform User Guide* in Designer's *Help Contents*.



3.7.5 Project Configuration

User projects created in Designer contain project-specific properties. Those projects containing the *Application Platform Core* project facet will include an Application Platform project property configuration.



3.7.5.1 *Project Version*

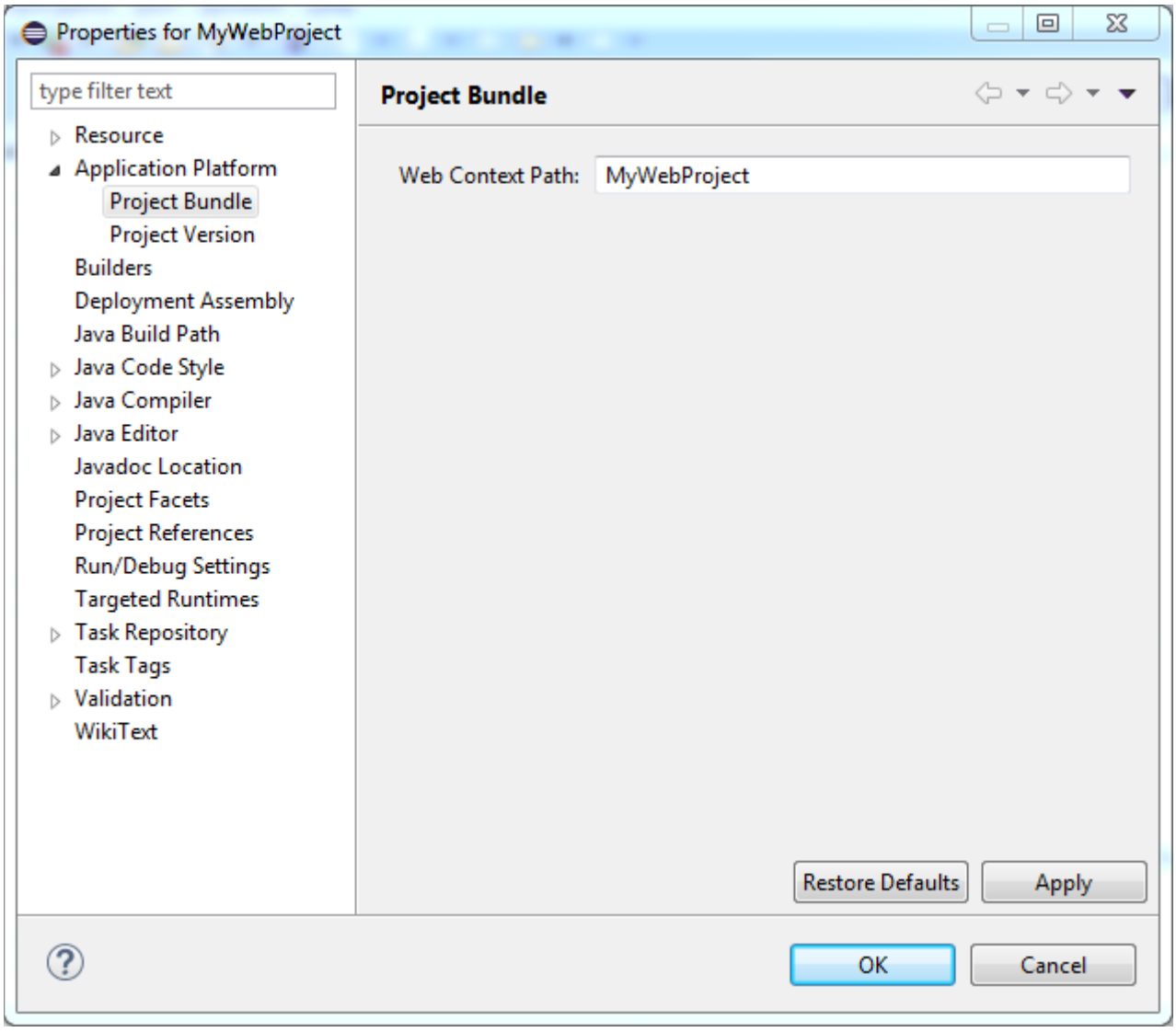
This version string must be valid for an OSGi bundle version. Specifically, it must include three numeric values separated by periods which are used to convey a version for the project's component. When a bundle is created for the project, its *Bundle-Version* manifest header will be set to the same value defined here.

Note: If an explicit manifest is created using the *Create Project Manifest* tool from the project context menu, the manifest must be updated (or recreated) if the version is changed to ensure the "Bundle-Version:" manifest header is up to date.

Important: To use this version property for projects deployed using Deployer, a project manifest must be explicitly created. The manifest must be included with other project files committed to source control, so it is available when using the Asset Builder Environment. This ensures the bundle produced by ABE will contain the expected Bundle-Version property.

3.7.5.2 Project Bundle

For web application projects created with the "Application Platform Web" project facet, a project's web context path may be changed after the project has been created.



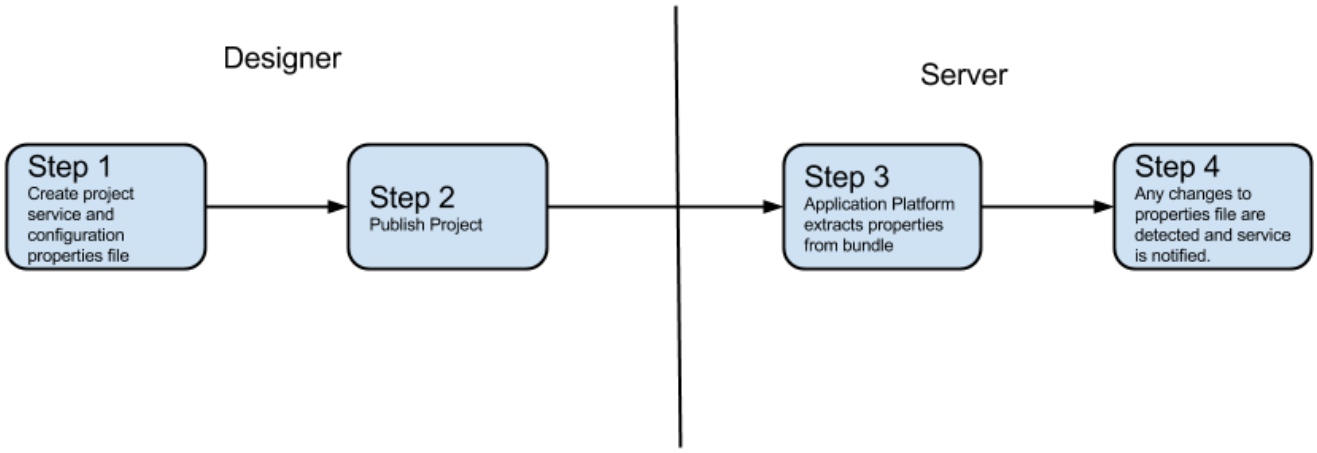
3.7.6 Customer Applications

Developed applications may include property files containing key-value pairs so these values may be configured in each server the application is deployed into. The Application Platform expects these key-value pairs to be implemented as properties files. This section explains how these properties files are created and installed into the server.

The following rules apply

- All properties files must be in the "*src/main/config*" folder for its project.
- Each property file must have a unique name so there is no filename collision once these files are deployed to the server.
- The file name should follow a reverse domain name convention. E.g. Company "*XYZ*" might have a com.xyz.demo.dataSource.properties file.
- File names beginning with "com.softwareag.*" are reserved for internal use. These files will not be deployed.
- Properties files may not be shared across projects since they are removed when a project is unpublished.

The diagram below illustrates the steps for managing configuration data while in Designer.



3.7.6.1 Step 1: Create Project

Use Designer to create a properties file in the "*src/main/config*" directory and implement a Java class to use these configuration properties.

Note: When a project is unpublished from the server using Designer's Servers view, the affected properties file will be removed from the server.

3.7.6.2 Step 2: Publish the Project

When the project is published from Designer, a bundle will be created and the contents from the "*src/main/config*" folder are included in the bundle. When the project bundle is created, a special "*AP-Bundle-ConfigFiles*" header is inserted into the manifest.

Note: Do not remove this property header from the bundle. This header is only produced when projects are published from Designer; The Asset Build Environment tool does not create this header when it produces a project bundle.

Important: Each time a project is published to the server, the configuration properties files contained in the project will overwrite any files that may reside on the server.

3.7.6.3 Step 3: Project Properties Extraction

When a project bundle is installed into the server, the Application Platform will inspect the bundle looking for the special "*AP-Bundle-ConfigFiles*" header. If found, it extras all listed property files and installs them into the server profile's directory for dynamic configuration – i.e. ${sag.install.dir}/profiles/${server instance}/configuration/com.softwareag.platform.config.propsloader/ directory.

3.7.6.4 Step 4: Notify Application Service of File Updates

After an application is published in the server, the server will respond to any subsequent edits to the properties file by notifying the managed service. For specific details, please consult the "Application Platform API and Programming Guide" in Designer's Help Contents.

3.8 Integration Server Features

Application Platform has features for exposing IS Java and Flow services to Application Platform projects. Java source file binding classes are code-generated to facilitate calling these IS services. The reverse scenario is also supported. Application Platform includes annotations that can be attached to methods. When projects containing these annotated methods are published to the Integration Server, IS service bindings are created which can be invoked in IS Flow services or executed from IS Java services.

3.8.1 Calling Integration Server Services from App Platform Projects

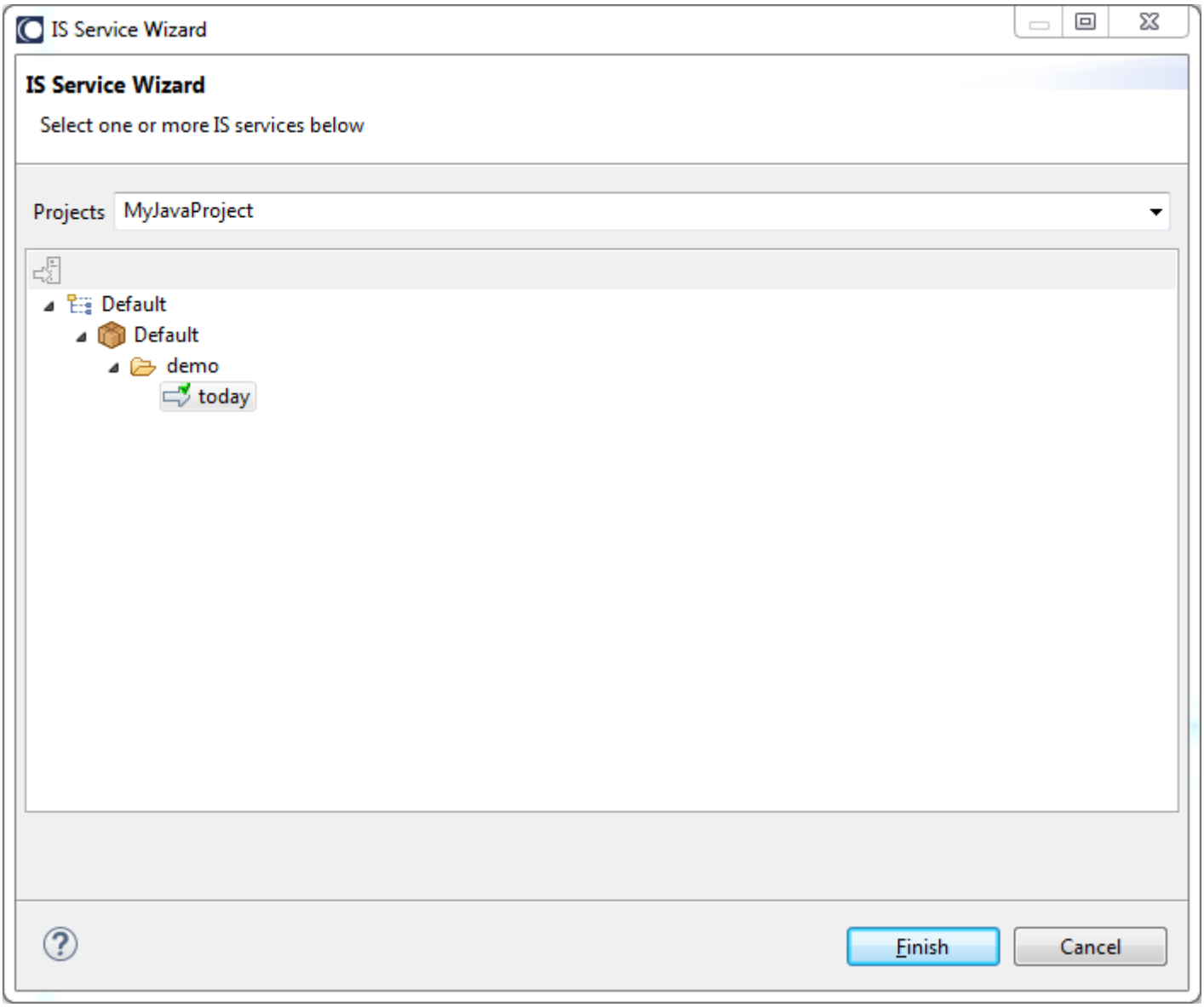
The following steps are necessary to create Java source files that are essentially client stubs used to invoke IS Java and Flow Services from an Application Platform project. For specific details, please consult the "Application Platform API and Programming Guide" in Designer's Help Contents.

3.8.1.1 IS Service Wizard

The IS Service Wizard is a tool for selecting the one or more IS services. It is used in the Application Platform perspective. One or more IS services may be selected. Once a destination project is selected and the user clicks the Finish button, a set of Java source files is generated. Only customer-developed services with valid IO specifications should be used. Only projects that have the IS Service Extensions project facet enabled may be selected.

Note: None of the Integration Server product services contained in packages that begin with Wm* are visible in the wizard. Therefore, no services are available for selection until at least one custom service has been created by the user.

For example, this screenshot shows one custom service in the demo package called "*today*".



Note: Control-Shift-Z key binding may be used to launch the wizard.

Two context menu actions are available when selecting a node in the tree view.

3.8.1.1.1 Connect to Server

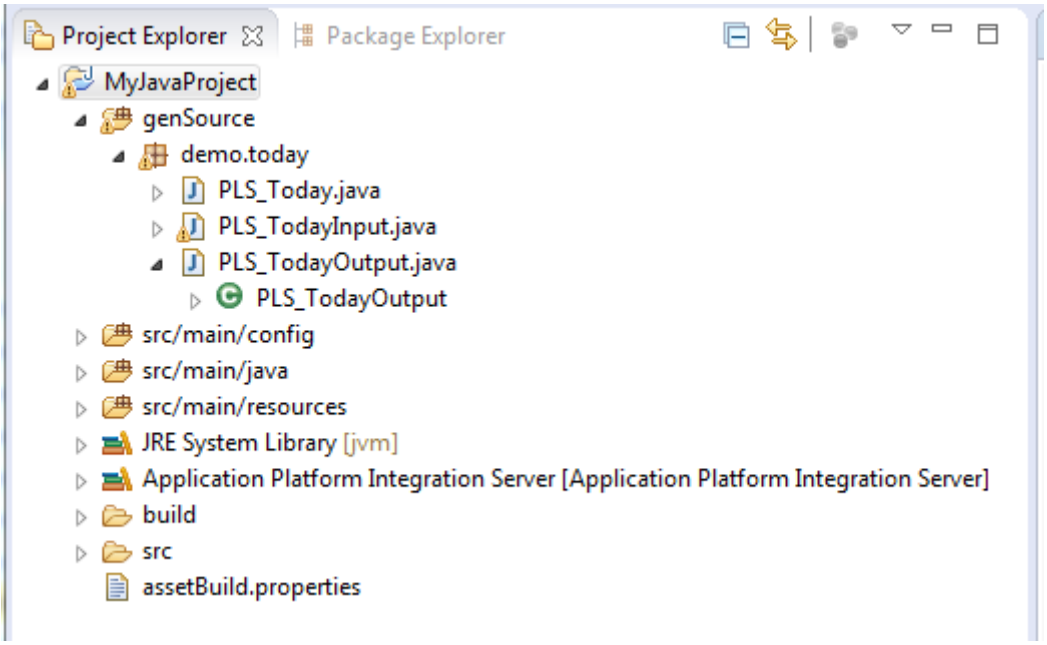
The "*Connect to Server*" context menu is enabled if Designer's Service Development perspective has not connected to server yet.

3.8.1.1.2 Refresh Tree Contents

The "*Refresh Tree Contents*" context menu is used to refresh the view to contain all created services and packages.

3.8.1.2 Generated Java Bindings

Pressing the Finish button completes the wizard and Java bindings are created for the selected services. The source files will be inserted into the source directory as defined for the IS Extensions project facet. The default location is "*genSource*". The package name is determined from the Integration Server service and its parent folder name(s). Each selected IS service has its own dedicated Java package to ensure there is no overlap with the generated input and output classes. Minimally, source files are created for the input, output and service invocation. Given the previous example, this is the package layout that is created for the generated source files.



Note: If deploying the project into production, the generated source directory should be set to "*src/main/java*". This restriction may be removed in future releases.

3.8.1.3 Coding considerations

There are some architectural differences between coding with Java and with IS services that should be kept in mind when using this Application Platform capability.

- Where Java is Object-Oriented, IS services are stateless operations on a pipeline. When a Java class is generated to represent an IS service, the class has a single method to represent service invocation. The IS pipeline is essentially a collection of name/value pairs, or map.
- Where Java methods are defined in classes which are in packages, IS services are defined in folders and in IS packages. Java packages and IS packages are different concepts. A class's Java package path uniquely represents the class in the Java class namespace. An IS package ('Default' in the example above) is a unit of packaging, but not part of the service namespace. In the IS namespace, the folder and service name uniquely identify a service ('demo:today' in the example above). The folder name in this example is very simple - in

general a folder name is a dot-separated list of words; for example, this.is.my.folder.name. It's not uncommon for IS folders to include capital letters, Java packages are almost always lowercase. Application Platform combines the IS folder and service names to create a Java package name ('demo.today' in the example above).

- Java and IS use different data types. Java's data type system is very rich, including primitive types and every class ever created. IS has a much smaller data type system. See the table below for data type mapping. IS supports String and Java primitive wrapper types, but complex structures IS services are typically modeled using Document data type. A Document is essentially a map where each element associates a name with a value. The values can be String, primitive wrapper or Document. So an IS Document with nested Document elements is very much like a map representing properties in a Java Bean. Application Platform takes advantage of this similarity. The generated input and output classes ('PLS_TodayInput' and 'PLS_TodayOutput' in the example above) are simple Java Beans with a property representing each input or output value. Is the IS service input signature includes a Document type then a Java (Bean) class is generated to represent the Document structure.

Java Data Type	IS Data Type
java.lang.String	String
Primitives; int, float, …	Object->Primitive Wrapper; Integer, Float, …
Primitive Wrappers; java.lang.Integer, java.lang.Float, …	Object->Primitive Wrapper; Integer, Float, …
java.util.Date	Object->Primitive Wrapper; Date
Java Bean class	Document Map of property name => property value, where String and primitive properties are represented as described above and other types are represented as nested Documents.

- Application Platform and IS use different class loaders, so object references are not transferred between them. Only String, Date, primitive wrappers and arrays of these elements have the same representation in IS and Java. More complex object structures are represented by Java Beans on the Java side and by IS Documents on the IS. The list of elements with similar representation includes byte array, so it is possible to pass serialized objects, but the Application Platform user must handle serialization and ensure that appropriate classes are available on both sides.
- Java and IS recognize different words as having special meaning. An IS service can have input parameter named 'class', but 'class' is a reserved word in Java. Also, a reserved word in Java may be a valid IS service or folder name. The AP code generation will prepend 'PLS_' or 'pls_' to generated class and property names to avoids some issues. But there may still be

situations where generated code does not compile properly. The simplest work-around for this situation is to use Flow mapping on the IS side to change parameter or service names.

- Java and IS recognize different sets of characters as having special meaning. An IS service and parameter names can use '@', '*' and other characters that are not allowed in Java class and variable names. The AP code generation does not currently attempt to detect or avoid this situation. As in the previous bullet, a simple work-around is to avoid the conflict by changing service and property names on the IS side.
- AP code generation relies on the IS service to have an explicit service signature defining all input and output elements. Such a signature is not required on the IS side, though recommended. If an IS service without signature must be called from Java and it's not possible to add a signature, then a Flow wrapper should be created that has an appropriate signature and invokes the service. The AP code generation can then work with this new Flow service.

3.9 Calling App Platform Services from Integration Server Services

Exposing Java methods to the Integration Server requires the use of annotations to mark the specific method(s) to expose. This section provides an overview of the required functional steps. For specific details, please consult the "Application Platform API and Programming Guide" in Designer's Help Contents.

3.9.1 Annotate a method

The first step is to mark the class and method with the necessary annotations. The @Service class annotation identifies the class as a service, so it can be included in the server's OSGi service registry. The @ExposeToIS class annotation provides additional details needed for the Integration Server. The @ExposedMethod method annotation identifies the method to be used when creating an Integration Server service.

```
@Service(name="OrdersService", interfaces={"com.softwareag.demo.orders.api.OrdersService"})
@ExposeToIS(packageName="OrdersService")
public class OrdersServiceImpl implements OrdersService {
```

Method

```
    @Override
    @ExposedMethod
    public String createReceiptEntry(LineItem inItem) {
```

3.9.2 Publish the Project

When the project's bundle is assembled, it will contain additional metadata to be used by the Integration Sever when creating IS service bindings.

3.9.3 Verify the IS package

Confirm the IS package described in the @ExposeToIS annotation exists, and it contains the proper service signatures.

3.9.4 Coding Considerations

There are some architectural differences between coding with Java and with IS services that should be kept in mind when using this Application Platform capability.

- IS services are stateless operations on a pipeline. There is no provision for holding references to Java objects in the IS pipeline. So, exposed operations should not depend on Java objects holding state.
- Application Platform and IS use different class loaders, so object references are not transferred between them. Java objects used in a method signature must be Java Beans. The IS services that are generated will include signatures that use Documents to represent the Java Bean objects.
- See the 'Coding considerations' section for 'Generated Java Bindings' above.

4 Production Activities

This section discusses tasks that occur after an application project is implemented, and it is time to deploy the project into servers which are downstream from the developer's environment. In this case, two items are discussed:

- Project deployment
- Project configuration

4.1 Project Deployment

Application Platform projects use the Software AG Deployer product to ensure reproducible builds of their developed applications are produced outside of Designer. This section presents a high-level explanation of the steps necessary to deploy a project. For precise details including configuration of properties files, please refer to the Deployer product documentation mentioned in the Deployer section. There are two steps to deploying a project using these tools.

4.1.1 Asset Build Environment

First, the Asset Build Environment (ABE) command line tool is used to create bundles from source for each of the projects. The Deployer documentation refers to these generated files as "*assets*". Before the tool can be executed, a set of properties files must be configured. There are two sets of properties files: one for the ABE tool, and one for each application project.

4.1.1.1 ABE Configuration

ABE has its own "*build.properties*" file in the "*${sag.install.home}/common/AssetBuildEnvironment/master_build*" directory. This file includes properties to control which products are included, the location of project source, etc.

4.1.1.2 User Project Configuration

For each of the customer's application projects, there is an "assetBuild.properties" file produced by Designer when the project is published. The file is created in the project's root folder. This file should be committed to source control with the rest of the project source files.

Property name	Value type	description	Required?
component.name	String	Name used for bundle file name, the Bundle-Name: and Bundle-SymboliName: manifest headers	Yes
component.type	String	For Application Platform, this value should always	Yes

		be "bundle"	
component.home	String	Path to project source location	No, ABE will assume a project with "component.name" in master_build/build.properties "build.source.projects"
component.dependencies	Comma delimited String	List of component names to be included on the classpath when building this project.	No
build.external.dir		Reserved for future use.	No
component.webcontext		Reserved for future use.	No, the web context path defaults to the project name, but it may be over-ridden via the WebContextPath: OSGi manifest header.
component.version		Reserved for future use.	No, Specify bundle version in the project's MANIFEST.MF file via BundleVersion: header.
component.src.dirs		Reserved for future use.	No, all source files must be under "src/main/java" directory.
component.dependencies.external		Reserved for future use.	No

For more details regarding ABE, please refer to the "*Deployer User's Guide*" cross-product document.

4.1.2 Shared Bundles

Currently, it is not possible to deploy bundles using ABE in the same manner as the Bundle Publisher in Designer. There are two alternatives available at this time.

4.1.2.1 Embedded in the Project

The current approach is to include bundles in the "lib" directory of the project so they will be part of that project bundle's classpath. This approach implies that every project requiring the dependency must include this bundle in its own project.

4.1.2.2 Manual Installation

Manually install a bundle into the target server. Please refer to third party OSGi documentation for more details.

4.1.3 Deploying Assets

Once assets are created using the ABE command-line tool, the Deployer tool is used to install assets into the target servers. For more details regarding Deployer, please refer to the "*Deployer User's Guide*" cross-product document.

4.2 Project Configuration

Application Platform projects may include configuration data with the projects which is included when the projects are published to the server. Once a project is on the server, those configuration values may be modified as needed.

4.2.1 Project Dynamic Configuration

The App Platform runtime supports dynamic configuration of project properties at runtime through the use of the OSGi ConfigurationAdmin service. In order to get the benefit of this feature, users must follow some guidelines when using configurable properties in their projects.

In traditional Java/JEE projects, configuration files such as properties files are loaded using the class/classloader of the currently executing method or thread. This implies that the files are present in the classpath of the running program and are accessible at runtime either with the project bundle that is published in the runtime or globally as part of the runtime container. The drawback of this approach is that the properties file contents cannot be dynamically updated through some external mechanism such as an admin interface without re-publishing the project bundle or restarting the runtime container.

To support dynamic updates to project properties, App Platform projects should use the following steps during development time.

1. Keep the .properties file in the src/main/config directory of the project
2. Name the properties file with a unique name – this name acts as a persistent identifier (PID) that identifies this resource. See the section Customer Applications for additional recommendations regarding the properties file.
3. Classes that need dynamic updates to this resource must:
    - Implement the OSGi org.osgi.service.cm.ManagedService interface and the associated updated (Map properties) callback method.
    - Publish this class as a managed service so that it can be notified about configuration file changes
        - This is done using the @Service annotation to publish the class as an OSGi service
        - The annotation must specify the org.osgi.service.cm.ManagedService type as one of the exported interfaces

The .properties file is packaged with the bundle being published in the container, but it is extracted and stored in the common configuration store in the installed runtime under this directory path: *${sag.install.dir}/profiles/IS_default/configuration/com.softwareag.platform.config.propsloader/*

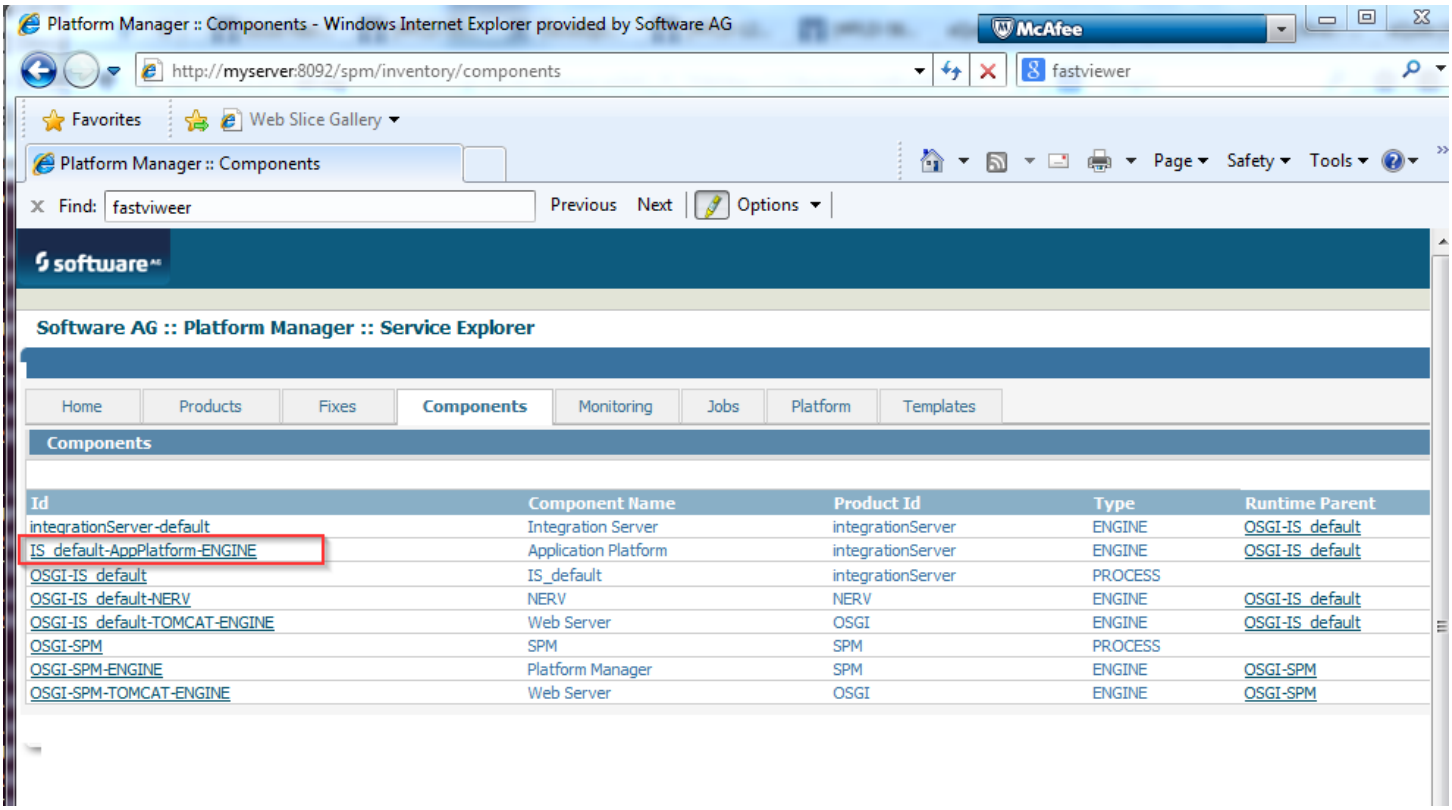
For more details, please consult the "Application Platform API and Programming Guide" in Designer's Help Contents.

4.2.2 Software AG Platform Manager (SPM)

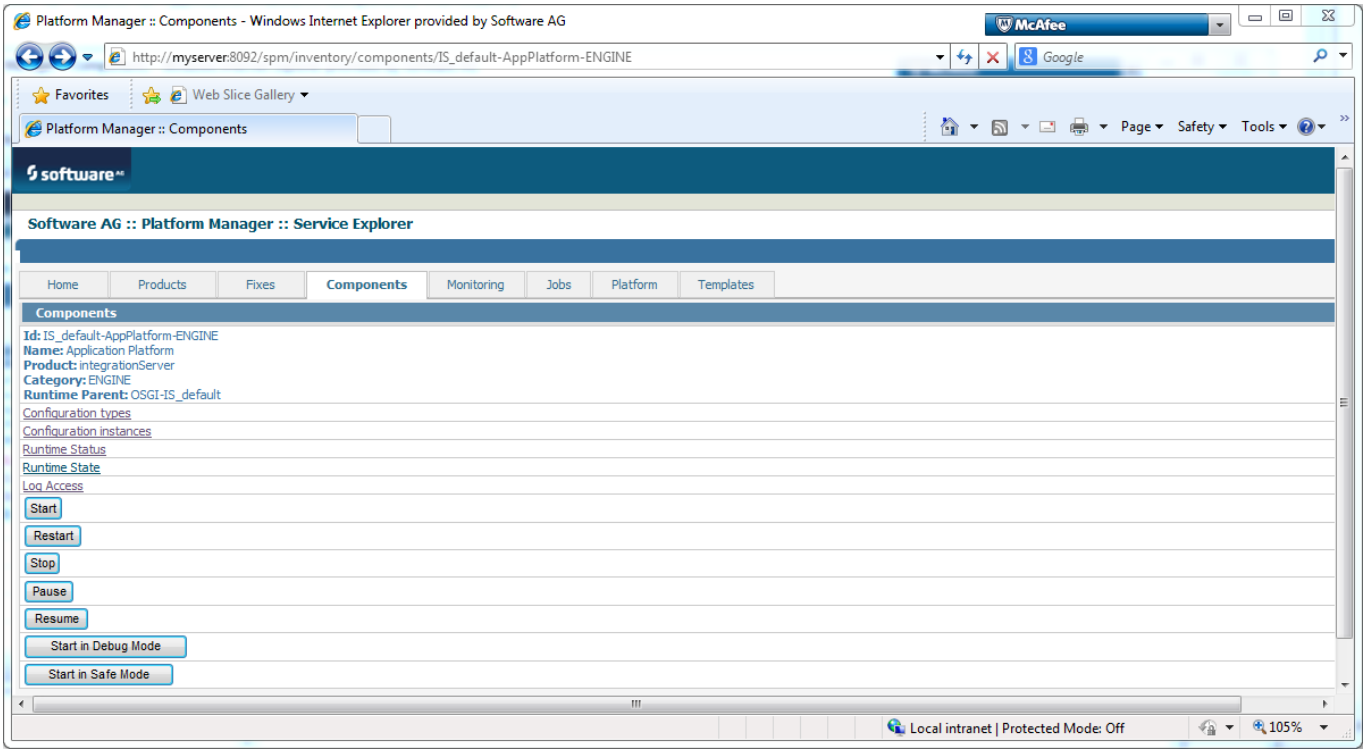
The platform manager is used to monitor and manage a product installation. The default URL is http://hostname:8092/spm/. This URL may be used to provide a read-only view of the Application Platform configuration details.

Note: The port number is determined by the product installer. This number will be different if the port is in use on the target system.

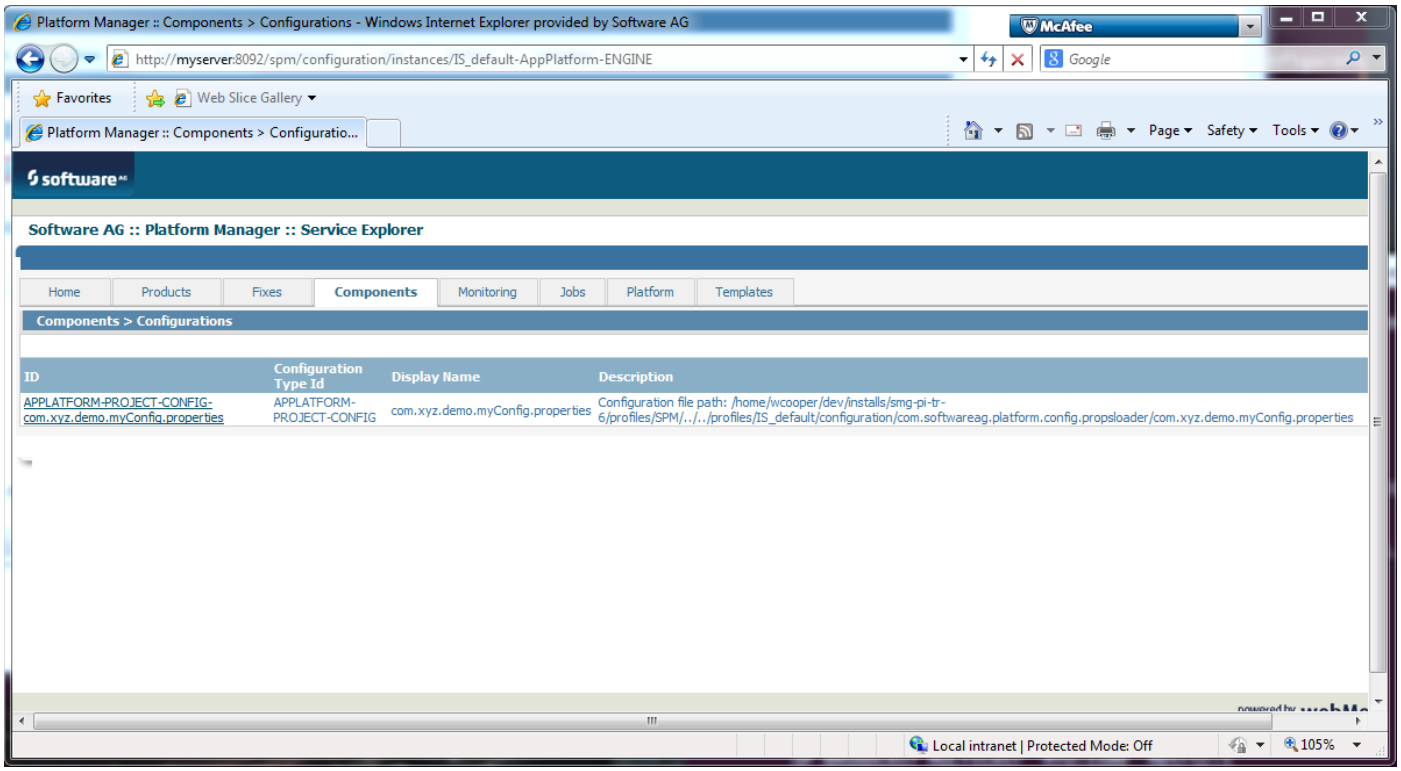
This screenshot shows an example of the Application Platform engine for the Integration Server's default instance. A project's configuration properties may be viewed by clicking through a sequence of pages starting with the *Component Id* hyperlink shown below.



Next, Click on the Configuration Instances hyperlink.



Clicking on the "*ID*" column hyperlink for the Application Platform component shown below will display the key-value pairs from the configuration file referenced in the "*Description*" column.



Note: After undeployment of a project, its configuration instance may still be cached in the Platform Manager. Append a "?refresh=true" query parameter to the URL to ensure the most current data is returned in the browser.

Please refer to product documentation found under the Platform Manager and Command Central section for more details.

4.2.3 Command Central Client Tools

The Command Central product includes command line interface and a server for hosting a web application used to access a configured group of products across one or more servers. For more details, please see the documentation referenced in the Platform Manager and Command Central section.

5 Troubleshooting

This section provides guidance to some of the more common issues. The sections are divided between Application Platform issues encountered in Designer versus the server.

5.1 Logging

5.1.1 Designer Log Files

The log file for Designer is *".log"* and may be found in a directory under the workspace. ${workspace}\.metadata\.log

5.1.2 Designer Trace Logging

Eclipse includes a tracing convention for capturing additional content in the event something goes wrong. Application Platform supports this convention too; however, some additional configuration steps are required. Please consult the Workbench Users Guide

5.1.3 Server Log Files

There are several log files produced in the server. For Integration Server's default instance, the path is: ${sag.install.dir}/profiles/IS_default/logs/. There are two files wrapper.log and sag-osgi.log.

5.1.4 Configure Server Debug Output

Additional debug output maybe captured by configuring appenders found in the server. For the Integration Server's default instance, the path is: ${sag.install.dir}/profiles/IS_default/configuration/logging/log_config.xml.

For example, to capture Application Platform debug messages, add this formatter to the log_config.xml and restart the server.

```
<logger name="com.softwareag.applatform.pls" additivity="true">
 <level value="debug" />
</logger>
```

For additional details, please refer to the product documentation under the Server section.

5.2 OSGi Console

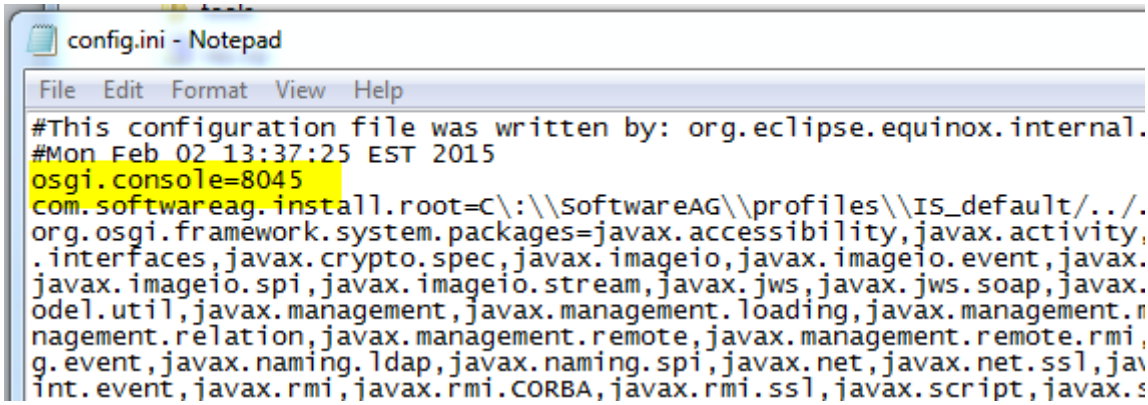
Designer includes a "Host OSGi" console in the Console view which may be used to examine the status of bundles installed in the Eclipse JVM. Similarly, it is possible to configure the server to permit access to an OSGi console too. The steps that follow illustrate how to setup an unauthenticated connection to the server in a development environment. It is not intended for productions systems.

Important: The OSGi console is an advanced feature. Proceed with caution when using this diagnostic tool.

5.2.1 Server Configuration

Before the OSGi console may be used to connect to the server, its OSGi configuration must be updated first. Edit the config.ini file for the server and add an unused port value for the key.

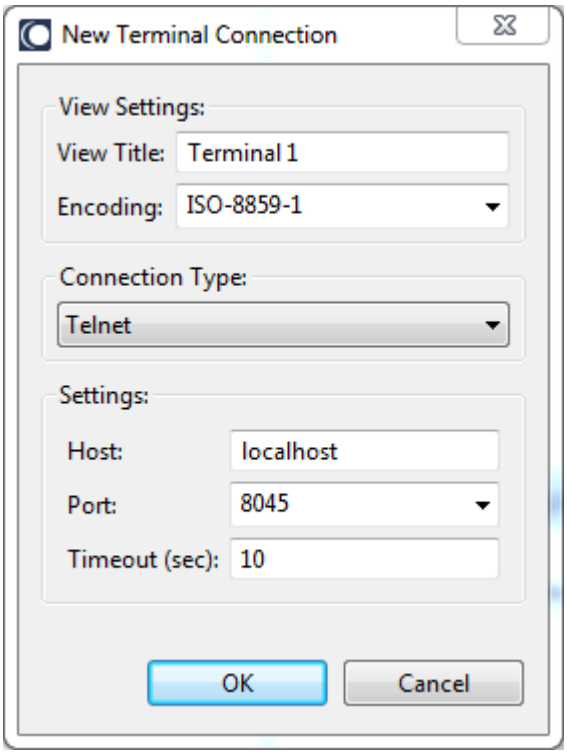
For example, c:\SoftwareAG\profiles\IS_default\configuration\config.ini



Note: If the server is running, it must be stopped and restarted before this change will take effect.

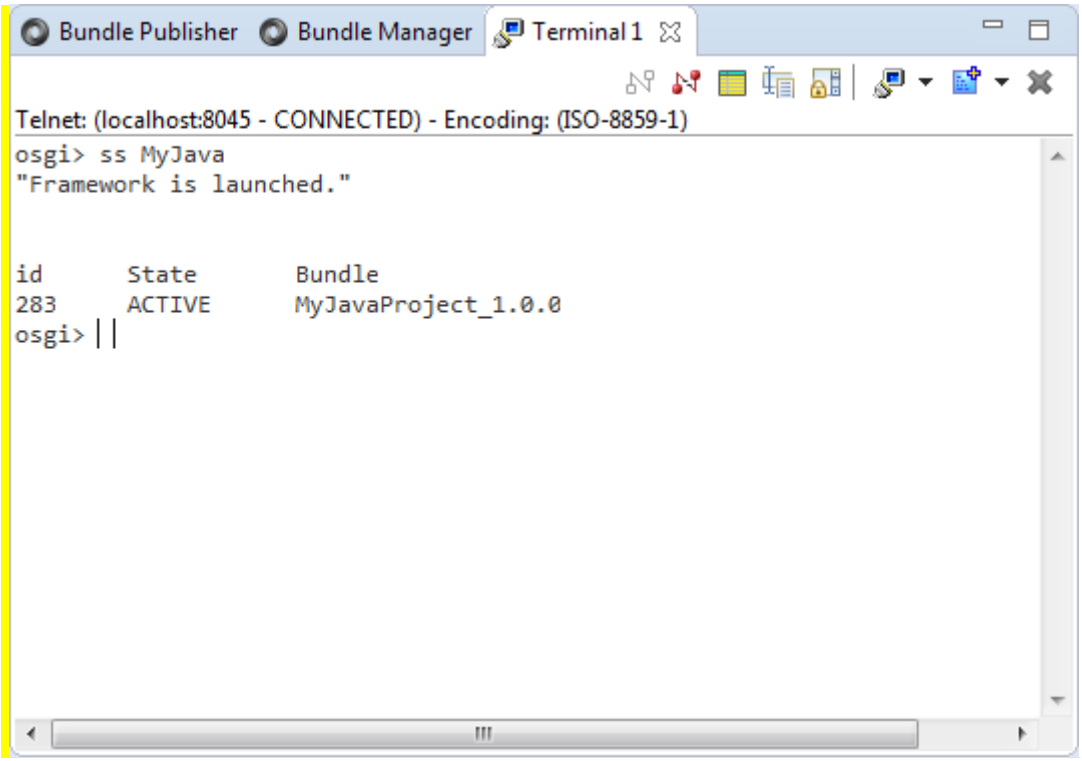
5.2.2 Terminal View Configuration

Once the server has been started, configure a view to connect to the port and press the Ok button.



5.2.3 OSGi Console

Click in the Terminal view and press the return key. An 'osgi>' prompt should appear. This screenshot shows the bundle state for a published project bundle. Help is available by typing "*help*" to get a list of the registered commands.



5.3 Server Views Problems

This section discusses the most common scenarios for publishing projects into the server.

5.3.1 Server is installed as a service

It is strongly advised that the server is installed as an "application" instead of a "service" when installing the product. This is because the service wrapper scripts will not start the server with the expected configuration. This can lead to a mismatched configuration between Designer and the server. For example, the service will be started without the JPDA debugging port configured and opened.

In the event of this situation, stopping the server and then restarting it from Designer should resolve the problem.

5.3.2 Server immediately fails to start

If the server state in Server views transitions to an error immediately after it is started, confirm the server start up script runs synchronously. Confirm the server's runtime environment

Note: Please confirm the following environment variable BLOCKING_SCRIPT is not set or set to yes – i.e. BLOCKING_SCRIPT=yes Consult the Integration Server product documentation for more details.

5.3.3 Server fails to start after timeout

Confirm the HTTP primary port number configured in the Servers view matches the port configured for the server instance.

Verify valid user credentials exist for the matching Integration Server port configuration found in Window/Preferences/Software AG/Integration Servers menu path.

5.4 Common Project Problems

This section covers some of the more common issues that can occur while creating and publishing project bundles.

5.4.1 Unable to Publish Web Projects

The WmTomcat package must be deleted or disabled to ensure no interference with the Common Tomcat Platform component. Leaving this package enabled can lead to failed publish attempts or projects whose web context fails to initialize.

5.4.2 Can't Add Project to Server

This can happen if the required Application Platform project facet(s) have not been included for the project. Projects cannot be added to the server unless they have the Application Platform core facet enabled. Be sure to use the Application Platform project wizards when building new projects.

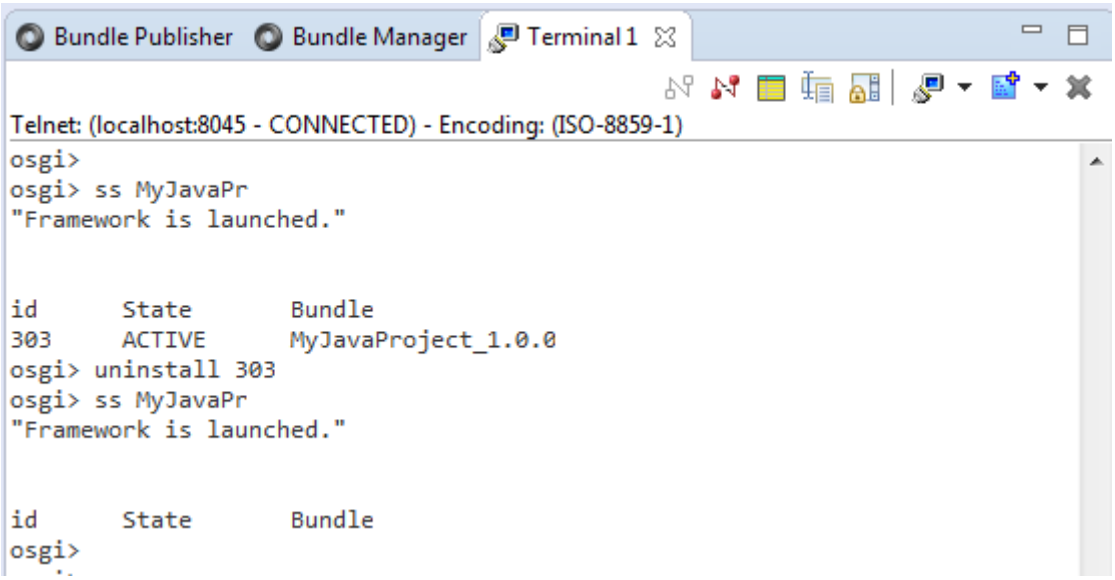
5.4.3 Unable to Create Bundle

A project containing Java source files in the default package is not currently supported. Ensure your source files have a qualified package name.

5.4.4 Manually Uninstall Bundle from Server

If a project is deleted from Designer while it is published to the server, the bundle will be orphaned. In this situation, manual steps must be taken to remove the bundle from the server. Two steps are required to complete this task while the server is started. For details about setting up an OSGi console to the server, please see OSGi Console section.

1. Delete the file from the repository directory. ${sag.install.home}/profiles/IS_default/workspace/app-platform/deployer/bundles/. This ensures the bundle is not re-deployed the next time the server is restarted.
2. Open an OSGi console to the server and uninstall the bundle using its bundle id.



5.4.5 Class Loader Issues in Published Projects

Careful inspection of the stack trace can provide helpful clues as to the nature of the problem. The following sections describe the most common scenarios.

5.4.5.1 ClassNotFoundException

A java.lang.ClassNotFoundException usually implies a situation where a class within the bundle fails to instantiate a class (e.g. Class.forName(classname)) for one of these scenarios:

- The class is not in the bundle raising the exception
- The class is not exported by any other bundle in the server,
- The class is exported from a bundle in the server; however, the bundle raising the exception does not import it

Usually, the 3rd scenario will be caught when the project is published as long as there is a source code reference to the class. Please see the Indirect Package Imports section for an explanation.

Note: Jars added to a project's classpath via its "lib" directory do not have its packages exported with the project bundle. This feature is intended as a means to extend a project bundle's classloader by including additional classes that are private to the project.

5.4.5.2 NoClassDefFoundError

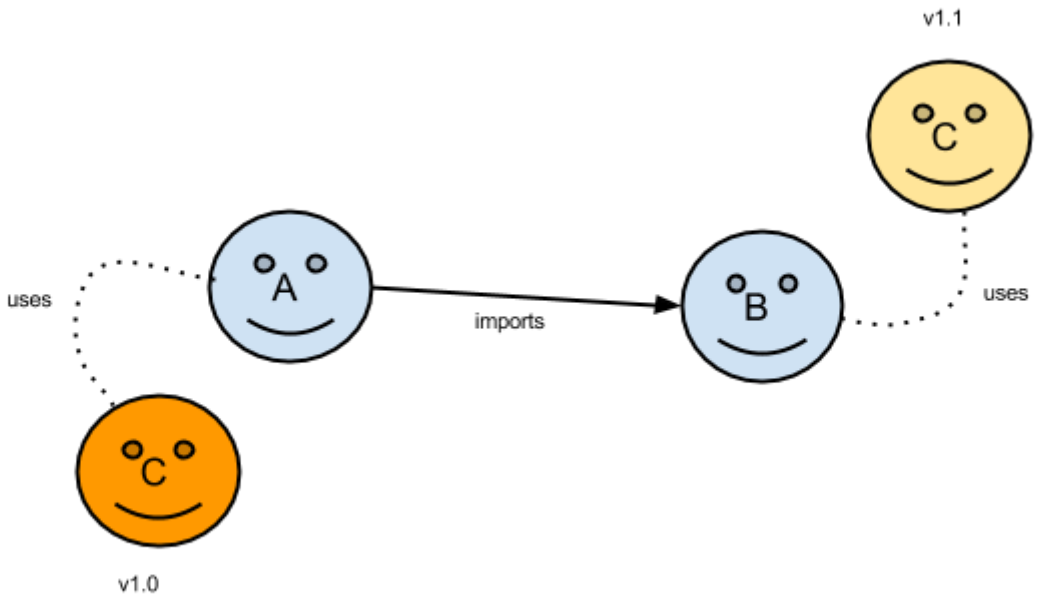
A java.lang.NoClassDefFoundError usually indicates that a loaded class in the bundle's classloader fails to reference another class while executing the loaded class.

5.4.5.3 LinkageError and ClassCastException

These errors are usually symptomatic of classloader pollution. Under normal circumstances, a collection of related bundles can be represented by a dependency graph based upon the chain of package imports and exports formed between these bundles. It's important to ensure that within a specific graph, only one instance of a class type is loaded and accessed across the graph. If there are multiple versions of the same class in the graph, then java.lang.LinkageError or java.lang.ClassCastException are produced.

OSGi provides the means to ensure multiple versions of class instances can be loaded in the JVM, but it's important to ensure one execution thread consistently uses the same class type. The OSGi "Export-Package:" header supports a "uses" directive to help provide clarity in the situation.

For example, in the diagram below, imagine bundle A imports a package from bundle B. One of the exported packages contains a class with a method signature that includes parameters that are found in bundle C. Meanwhile, bundle A has dependencies to another version of bundle C which leads to an invalid classloader graph.



Note: If jars are added to a project's classpath via its "lib" directory, care should be taken to ensure these classes are not already exported by another bundle for the reasons mentioned above.

5.4.6 References to Local Resources

Traditional Java programming techniques that rely upon access to metadata files (e.g. Java service provider), Thread context classloader, etc. may not be remotely referenced across bundle boundaries.

5.4.7 Unable to Publish Any Project Bundle

The server and designer must be co-located under the same root installation directory. The Runtime Environment configuration is stored in a file under the workspace. This can lead to confusion if multiple workspaces are used with multiple installations of Application Platform. Confirm the runtime environment's directory path is correct for the current workspace. For example, if using Designer installed under c:\SoftwareAG with workspace C:\dev\workspace_98\, make sure the Runtime Environment configuration indicates C:\SoftwareAG\ for the installation home.

See the Server Runtime Environment section for more details.

5.5 Miscellaneous

This section covers a variety of issues.

5.5.1 Configuring an Eclipse project for Application Platform

This section provides the minimum steps which are required for importing an Eclipse project that was not created using an Application Platform project wizard. The steps are defined for the two most likely scenarios.

Note: The Application Platform project facets may be safely unselected and re-selected for a project. No files are deleted when uninstalling one of these project facets.

5.5.1.1 Java Perspective Project Wizard

This section covers the steps for a project that was created using the basic "Java Project" project wizard from the Java perspective.

1. Configure the project to use project facets for Application Platform. This can be done by performing the following steps
    a. Select the project in the Package or Project Explorer and invoke the context menu by right-clicking and selected "Properties"
    b. Select "*Project Facets*" from the Project preferences dialog.
    c. Click the "*Convert to faceted form…*" link.
    d. Ensure the Java project facet is selected for the appropriate version

e. Click the Software AG Application Platform group and select the *"Application Platform Core"* facet.

f. Click the server extensions facet if necessary. For example, "*Integration Server Extensions"* should be selected if using the IS Service Wizard.

g. Click the "Further configuration available…" link to provide additional configuration if necessary. See the Project Facets section for more details.

2. Confirm the following actions were performed on the project after applying the facet changes.

a. The source code folder "src" moved to "/src/main/java" and selected in project's "*Build Paths*" dialog.

b. A "*src/main/resources*" folder was created and selected in the project's "*Build Paths*" dialog.

c. A "*lib/*" folder was created.

d. A "src/main/config" folder was created.

5.5.1.2 Dynamic Web Project Wizard

This section covers the steps for a Servlet-based project that was created using the "Dynamic Web Project".

1. Edit the org.eclipse.wst.common.project.facet.core.xml file by removing the fixed element for the "jst.web" facet. This file is in the ".settings" folder under the project's directory in the Eclipse workspace.
2. Refresh the project in the Package or Project Explorer.
3. Configure the project to use project facets for Application Platform. This can be done by performing the following steps

a. Select the project in the Package or Project Explorer and invoke the context menu by right-clicking and selected "Properties"

b. Select "*Project Facets*" from the Project preferences dialog.

c. Ensure the Java project facet is selected for the appropriate version

d. Uncheck the "*Dynamic Web Module*" facet. (The fixed element in step 1 must be removed before this can be accomplished.)

e. Click the Software AG Application Platform group and select the *"Application Platform Core"* and "*Application Platform Web*" facets.

f. Click the server extensions facet if necessary. For example, "*Integration Server Extensions"* should be selected if using the IS Service Wizard.

g. Click the "Further configuration available…" link to provide additional configuration if necessary.

4. Confirm the following actions were performed on the project after applying the facet changes.

a. The source code folder "src" moved to "/src/main/java" and the new location is selected in Project's "*Build Paths*" dialog.

b. A "*src/main/resources*" folder was created and selected in the project's "*Build Paths*" dialog.

c. A "*lib/*" folder was created.

d. A "src/main/webapp/WEB-INF" folder was created.

e. A "src/main/config" folder was created.

5. Use Designer to move the servlet content (e.g. jsp, css, javascript, html, etc.) from its current location to the "src/main/webapp/…" path. For example, if the dynamic web project "A" had its images under / *"{A}/WebContent/images/…",* then the images folder would move to *"{A}/src/main/webapp/images/…".*

6. Repeat Step 5 until all of the web content has been re-located under the "src/main/webapp" directory path.

