

webMethods Application Platform User's Guide

Version 10.11

October 2021

This document applies to webMethods Application Platform 10.11 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.

Copyright © 2014-2021 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <https://softwareag.com/licenses/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <https://softwareag.com/licenses/> and/or in the root installation directory of the licensed product(s).

Document ID: AP-UG-1011-20211015

Table of Contents

About this Guide.....	5
Document Conventions.....	6
Online Information and Support.....	6
Data Protection.....	7
Application Platform Deprecation.....	9
1 About Application Platform.....	11
Architecture and Components.....	12
Publishing and Deploying Bundles.....	15
2 Developing with Application Platform in Designer.....	17
Getting Started with Application Platform Development.....	18
About the Application Platform Perspective.....	23
About Creating Application Platform Projects.....	26
Configuring a Designer Project for Application Platform.....	35
About Developing Web Applications for Integration Server.....	37
Developing Web Archive (WAR) Projects.....	41
Developing Web Application Bundle (WAB) Projects.....	41
About Adding Single Sign-on Authentication in Application Platform Projects.....	41
About the Application Platform Integration Test Framework.....	45
Managing Servers.....	47
About Publishing Projects.....	60
About Viewing Dependency Graphs.....	64
About Managing Project Dependencies.....	69
Configuring Application Platform.....	75
About Using Services in Application Platform.....	82
Application Platform Tutorial.....	90
3 Working with Application Platform Projects.....	93
About Deploying Projects.....	94
About Configuring Published Projects.....	97
4 Administering Application Platform Using Command Central.....	99
Managing Application Platform Projects Using Command Central.....	100
Monitoring KPI Data for WAR Projects Using Command Central.....	100
5 Diagnostics and Troubleshooting.....	103
Useful Logs for Application Platform.....	104
Increasing Tomcat Debug Logging.....	104
Using Log4j in WAR projects.....	106
JSP Validation in WmAppPlat.....	107

Diagnosing Bundles with the OSGi Console.....	107
Considerations When Publishing Projects to Servers.....	107
Common Project Issues.....	108
A Differences Between WmTomcat and WmAppPlat.....	111

About this Guide

- Document Conventions 6
- Online Information and Support 6
- Data Protection 7

This guide provides information about working with webMethods Application Platform aimed at application developers. It explains common tasks, such as building Java applications using Application Platform projects in Software AG Designer, packaging the applications in OSGi bundles, and deploying them to an Integration Server instance.

To use this guide effectively, you should be familiar with webMethods Integration Server and Software AG Designer. You should understand the concepts and procedures in the *webMethods Integration Server Administrator's Guide* and the *Software AG Designer Online Help*. You should also have basic knowledge of OSGi and its concepts.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies service names and locations in the format <i>folder.subfolder.service</i> , APIs, Java classes, methods, properties.
<i>Italic</i>	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
Monospace font	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
	Separates two mutually exclusive choices in a syntax line. Type one of these choices. 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 (...).

Online Information and Support

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at <https://documentation.softwareag.com>.

Software AG Empower Product Support Website

If you do not yet have an account for Empower, send an email to empower@softwareag.com with your name, company, and company email address and request an account.

Once you have an account, you can open Support Incidents online via the eService section of Empower at <https://empower.softwareag.com/>.

You can find product information on the Software AG Empower Product Support website at <https://empower.softwareag.com>.

To submit feature/enhancement requests, get information about product availability, and download products, go to [Products](#).

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the [Knowledge Center](#).

If you have any questions, you can find a local or toll-free number for your country in our Global Support Contact Directory at https://empower.softwareag.com/public_directory.aspx and give us a call.

Software AG Tech Community

You can find documentation and other technical information on the Software AG Tech Community website at <https://techcommunity.softwareag.com>. You can:

- Access product documentation, if you have Tech Community credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

Application Platform Deprecation

webMethods Application Platform is deprecated. To build Java services for your new development projects, use webMethods Service Development instead. For more information, see *webMethods Service Development Help*.

1 About Application Platform

■ Architecture and Components	12
■ Publishing and Deploying Bundles	15

webMethods Application Platform complements the webMethods product line by allowing you to create custom business applications. You can use Application Platform together with other Software AG products to create entire business solutions. Application Platform provides a user friendly way for building custom business logic.

Application Platform enables you to create custom business applications in a Java-friendly way. It is based on OSGi and provides development tools and run-time components for building Java applications using Software AG Designer. You can use standard Designer tools for debugging and you can directly deploy your project bundles to a webMethods runtime without using a third-party application server. Project bundles are deployed to webMethods Integration Server in exactly the same way, as in other webMethods products. You can also integrate your Application Platform projects with Integration Server (IS) services.

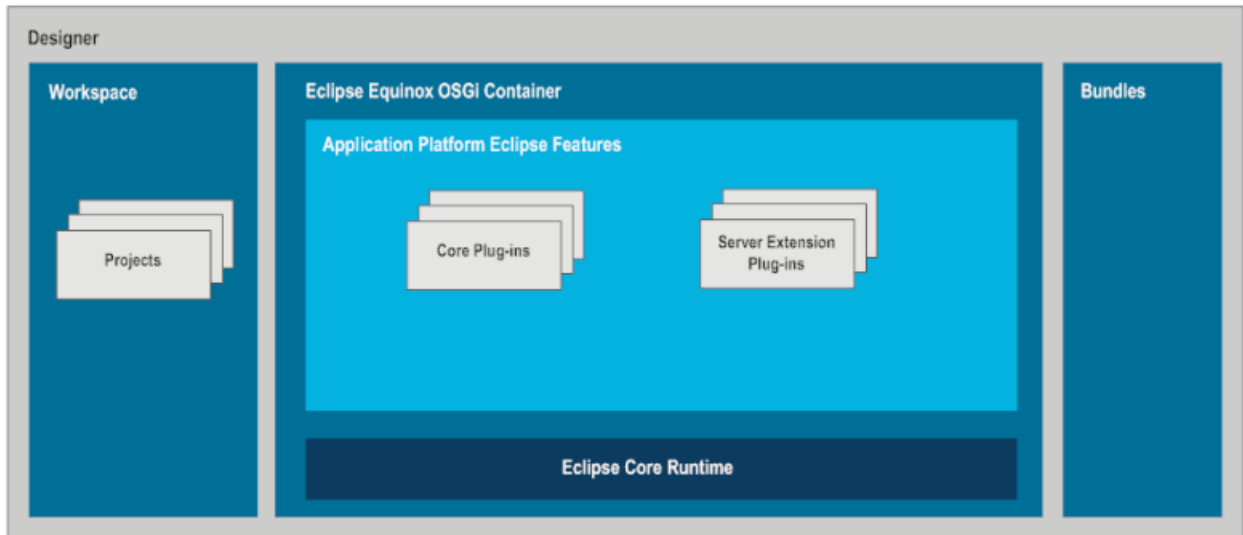
With Application Platform you can complete the following tasks:

1. Develop business logic applications in Designer by using standard POJOs and Spring Beans.
2. Integrate your applications with Enterprise Service Bus (ESB) services through bi-directional invocation.
3. Deploy and run the applications on your existing webMethods servers alongside your existing webMethods assets.
4. Expose your applications to users in various ways by using web applications.

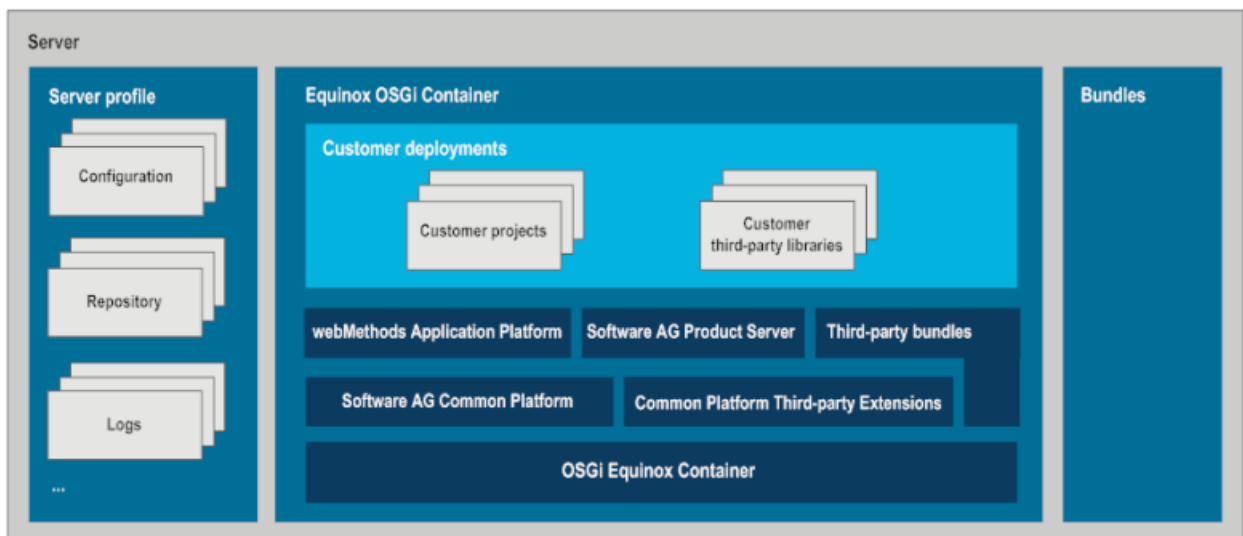
Architecture and Components

The Software AG Designer architecture consists of a development layer, which resides in Designer, and a server layer, which is responsible for the runtime operations.

The following diagram displays the Software AG Designer development components, which include the Designer workspace to the left, the Eclipse Equinox OSGi container in the middle, and the project bundles component to the right. The workspace contains the project folders. The Eclipse Equinox OSGi container has the Application Platform Eclipse features in the upper section, containing core plug-ins and server extension plug-ins, and the Eclipse core runtime component in the lower section.



The following diagram displays the server components, which include the server profile to the left, the Equinox OSGi container in the middle, and the project bundles to the right. The server profile contains configuration files, project repositories, and log files. The Equinox OSGi container section consists of customer deployments in the upper part, containing core plug-ins and server extension plug-ins, and the following components in the lower part: Application Platform, Software AG Product Server, third-party bundles, Software AG Common Platform, Common Platform third-party extensions, and the OSGi Equinox container.



For further explanation, see the sections for each component that follow the diagram.

Software AG Common Platform

The architecture of Application Platform is based on Software AG Common Platform. Common Platform is OSGi-based and it enables you to dynamically construct executable instances of various products.

In Common Platform, a deployment module is typically a Java jar file, called a *bundle*, that contains a `META-INF/manifest.mf` file with additional headers. When you install bundles to a server, the OSGi container of the server uses the metadata, provided in the additional headers. The OSGi container implementation used in Application Platform-supported servers is Eclipse Equinox.

With Application Platform, you create and install OSGi bundles to a server. Third-party jar files should also be OSGi bundles. Alternatively, you can use Application Platform to create bundles from simple jar files. You can also embed plain jar files by placing them in the Application Platform project's `lib` folder.

For information about creating plain jar files, see [“Including Non-OSGi Jars in Projects” on page 35](#).

Software AG Designer

Application Platform uses Designer as an integrated development environment for building components. In Designer, you use the Application Platform perspective and the following functions:

- Project wizards for creating Java and web applications
- Integration with server tools for publishing and debugging projects to the server
- Dialog wizards for creating Java bindings to server components
- Options for customizing your perspective and views
- Various utilities for developing projects on the Software AG Common Platform

For more information about the functions of Designer, see *Working with Software AG Designer*.

For more information about developing services in Designer, see *webMethods Service Development Help*.

Software AG Servers

Application Platform projects are published or deployed to a Software AG server, that meets the following requirements:

- The server is based on OSGi runtime containers.
- The server supports Software AG Common Platform.
- The server is managed by a set of scripts for cross-cutting control. Examples of such scripts include:
 - logging scripts
 - configuration scripts
 - lifecycle scripts

For more information about Software AG servers, see *Software AG Infrastructure Administrator's Guide*.

webMethods Deployer

When you have completed your projects in Application Platform, you can deploy them to the required servers by using command line scripts in webMethods Deployer. Deployed projects are built and packaged as assets by using webMethods Asset Build Environment. You can deploy all assets to one or more target systems with Deployer.

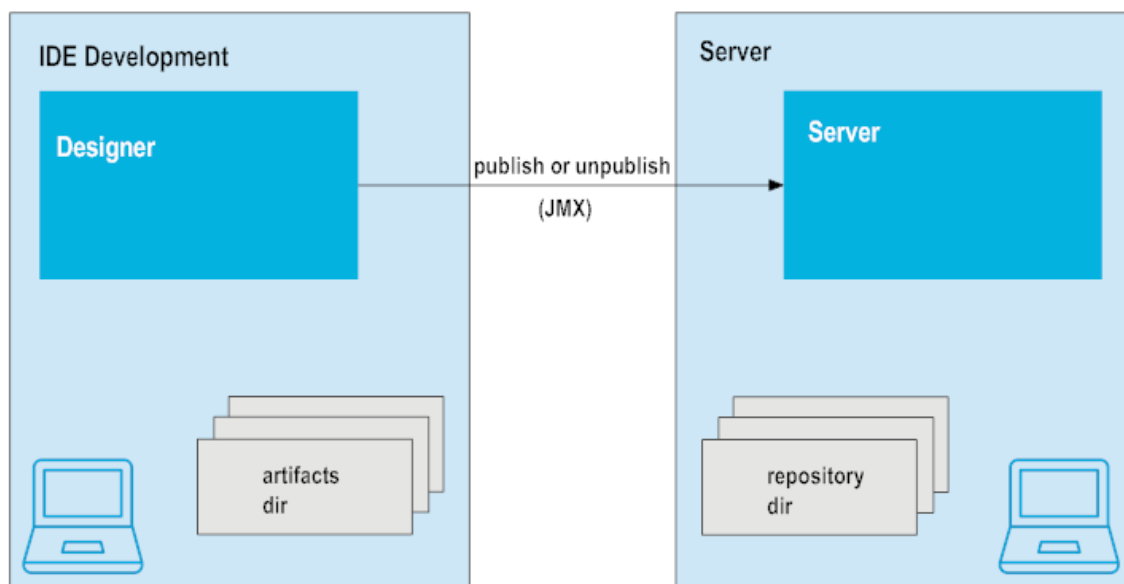
For more information about Deployer and Asset Build Environment, see the *webMethods Deployer User's Guide*.

Publishing and Deploying Bundles

In this guide we differentiate between the terms *publish* and *deploy*. Project bundles are published when the deployment activities are performed from the integrated development environment, which is Designer for Application Platform, to webMethods Integration Server. Project bundles are deployed when the deployment activities are performed outside of Designer. When Application Platform projects are deployed, project assets found in an asset build environment repository are deployed to Integration Server via webMethods Deployer.

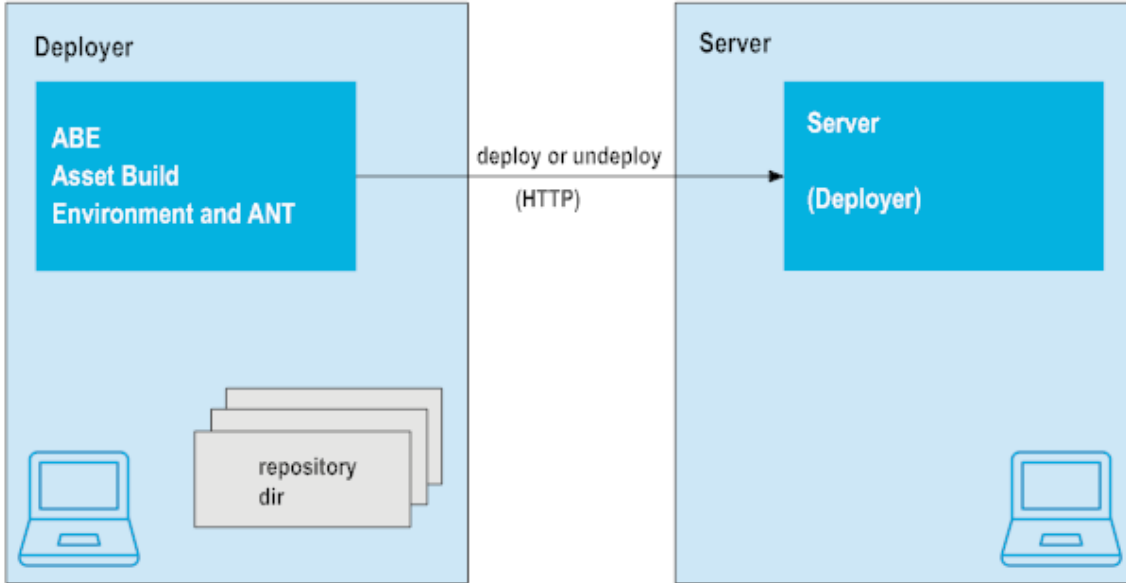
For information about deploying assets using Deployer, see [“About Deploying Projects” on page 94](#).

When you publish project bundles in Designer, your bundles are published to a server on the same physical machine or a remote server. Each component that is involved in the publishing process requires a Java Virtual Machine (JVM). The following diagram illustrates the work flow for publishing bundles to a dedicated server. You create your projects in Designer, which contains the artifacts directory for your projects. Projects are published or unpublished from the server through a Java Management Extensions (JMX) protocol. When you publish a project to the server, it is stored in a dedicated repository directory on the server.



When you deploy project bundles using Asset Build Environment (ABE) and Deployer, your bundles are published to a remote server, which is on a different physical machine. Each component

that is involved in the deployment process also requires a JVM. The following diagram illustrates the work flow for deploying bundles using Deployer. Deployer uses project bundles, created through the Asset Build Environment. The required files are stored in a repository directory on your local machine, which Deployer uses to deploy or undeploy projects on the server through HTTP.



2 Developing with Application Platform in Designer

■ Getting Started with Application Platform Development	18
■ About the Application Platform Perspective	23
■ About Creating Application Platform Projects	26
■ Configuring a Designer Project for Application Platform	35
■ About Developing Web Applications for Integration Server	37
■ Developing Web Archive (WAR) Projects	41
■ Developing Web Application Bundle (WAB) Projects	41
■ About Adding Single Sign-on Authentication in Application Platform Projects	41
■ About the Application Platform Integration Test Framework	45
■ Managing Servers	47
■ About Publishing Projects	60
■ About Viewing Dependency Graphs	64
■ About Managing Project Dependencies	69
■ Configuring Application Platform	75
■ About Using Services in Application Platform	82
■ Application Platform Tutorial	90

This topic describes the features added to Designer to support developing with Application Platform.

Getting Started with Application Platform Development

The following sections describe tasks that you should perform after you install Application Platform in order to start building projects.

Opening the Application Platform Perspective

Application Platform has a dedicated Designer perspective. This perspective contains the basic views you need in order to develop applications.

> To open the Application Platform perspective

1. In Designer, go to **Window** menu, select **Perspective > Open Perspective**, and then click **Other**.
2. Click **App Platform** and then click **OK**.

If you have not created a runtime environment for Application Platform, a warning message will be displayed after opening the App Platform perspective.

After you open the App Platform perspective for the first time, it is cached in the upper right corner of Designer for quick access.

3. Optionally, if a warning message is displayed after you execute step 2, click **Yes** and configure a runtime environment.

Designer redirects you to the App Platform Runtime configuration view.

Adding a Server Runtime Environment

When you first install Application Platform, you must add a server runtime for Application Platform, so that your projects can reference their runtime container. For the runtime configuration, use an absolute path to the product installation. Runtime containers are Designer configuration elements that define a set of product libraries that are included in project classpaths.

> To add a server runtime environment for Application Platform

1. In Designer, go to **Window** menu and click **Preferences**.
2. In the Preferences dialog box, click **Server**, and then click **Runtime Environments**.
3. In the Server Runtime Environments dialog box click **Add**.

4. In the New Server Runtime Environment dialog box, select an Application Platform server, and then click **Next**.

The available server is Integration Server, so you can select **Application Platform Integration Server**, or **Application Platform Integration Server (Remote)**.

5. In the **Designer installation root directory** field, enter the path to the Software AG installation folder.

Depending on the type of server you are using, keep in mind the following:

- If you are configuring Integration Server runtime with a local Integration Server, the installation folder must reside in a *Software AG_directory*, which contains a *profiles* directory.
- If you are configuring Integration Server runtime with a remote Integration Server, you must also have an Integration Server instance installed on the machine where Designer is installed. In this way Designer can access the server runtime's libraries when building projects.

Important:

The installation root directory of Designer is stored in the Eclipse workspace metadata area. If you install another instance of Designer on the same machine, you must not use the same workspace directory. Using the same workspace directory for more than one instance of Designer can lead to errors, since both instances will share the same runtime configuration and will communicate to the same server.

6. Click **Finish**.

After adding the server, you must configure it for your Application Platform projects.

For more information about configuring Integration Server, see [“Configuring Integration Server for Application Platform Projects” on page 47](#).

Creating a Server Definition

After you configure a runtime environment, you must create a server definition in order to publish projects. Application Platform supports webMethods Integration Server.

When you have a runtime environment and a server configuration, you will be able to manage the development server in Application Platform. For information about using the server tools, see the *Web Tools Platform User Guide* in *Software AG Designer Online Help*.

Note:

As prompted during the server installation, Software AG recommends that you install the server as an application, as opposed to a service. When you start a server as a service, the server does not run in debug mode. This prevents Designer from remotely debugging the server instance.

For information about the issues, related to installing a server as a service, see [“Considerations When Publishing Projects to Servers” on page 107](#).

Creating an Integration Server Definition

This section describes the steps for creating an Integration Server definition in Application Platform.

➤ To create an Integration Server definition

1. Go to the Servers view at the bottom of the App Platform perspective.
2. Click the link for creating a new server or right-click anywhere in the Servers view, select **New**, and click **Server**.
3. In the Define a New Server page of the New Server dialog box, specify values in the provided fields.

The following table describes what you have to specify for each setting.

For this setting	Specify
Select the server type	The type of server to be added. For Integration Server select webMethods Integration Server under the Software AG directory.
Server's host name	The host name or address of the Integration Server, to which you publish projects. If you have created a runtime with a local Integration Server, keep the value of this field to <code>localhost</code> . Default: <code>localhost</code>
Server name	The name of the Integration Server, to which you publish projects. Default: <code>webMethods Integration Server at host_name</code>
Server runtime environment	The server runtime environment to be used by the Integration Server. All server runtime environments that you have added for Application Platform are listed here.

4. Click **Next**.
5. If you have changed the server properties during installation, specify the values for the provided settings. Otherwise, keep the default values.

The following table describes what you have to specify for each setting.

For this setting	Specify
Instance name for Integration Server	The instance name of the Integration Server on the specified address. Integration Server allows multiple instances on the same machine. The default value of this field is <code>default</code> .
Server Port	The port number of the primary port of Integration Server.

For this setting	Specify
	Default: 5555
Server Debug Port	The port number that is used by Integration Server when you debug an Application Platform project in Designer. Default: 9191
Server JMX RMI Port	The port number that is used for monitoring Integration Server remotely using a JMX agent. Default: 8075
Server Connection Mode	The connection mode to be used when connecting to the Integration Server. The following modes are available: <ul style="list-style-type: none"> ■ Debug - Default. This mode depends on your runtime environment, as follows: <ul style="list-style-type: none"> ■ If you have configured a runtime environment with a local Integration Server, Designer will automatically start debugging the servers in the Servers view. For example, if you restart Designer, the server instance in the Servers view will automatically start debugging. ■ If you have configured a runtime environment with a remote Integration Server, you must start the server in debug mode from the machine, where the server is installed. ■ No Action - In this mode Designer will not synchronize the state of the Servers view with the server. If Designer is started and a server is running, the Servers view will indicate the server is stopped. In this case you must execute the Start or Debug action in the Servers view. Additionally, if the status of the server changes while Designer is still running, the change will not be indicated in Designer. ■ Run - In this mode Designer will automatically set the server status to “started” and you will not be able to debug applications remotely, while Designer is connected to the server.

6. Click **Next**.
7. To add the projects that you want to configure on the server, select the name of a project in the **Available** field and click **Add**.
8. Click **Finish**.

Configuring a Server for Publishing Bundles

If you want to publish your Application Platform projects to Integration Server, you must ensure a functional environment for publishing bundles to the server. Verify that the Software AG Runtime component is available for Integration Server .

For detailed steps, see [“Verifying That Software AG Runtime Is Available” on page 22](#).

Verifying That Software AG Runtime Is Available

Application Platform uses a Software AG Common Platform component called Software AG Runtime. This component is enabled by default after installing Application Platform.

Software AG Runtime uses the following default ports for Integration Server:

- HTTP: 8072
- HTTPS: 8074

➤ To verify that Software AG Runtime is available after you install Application Platform

1. Depending on the server type you are using, start the Integration Server instance.
2. In a web browser enter `http://localhost:8072`.

If you can successfully load the server log-on page, this indicates that Software AG Runtime is available.

Performing Optional Configurations

This section describes optional configurations, which you can perform after installing Application Platform.

Disabling Natural Language Support (NLS) Warnings in Designer

Designer produces warning messages for localized messages (NLS messages). NLS warning messages do not indicate installation problems, but that a localized message is not used. NLS warning messages are in the following format:

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

Displaying such messages can lead to situations when too many log messages are generated.

➤ To disable the generation of NLS messages

1. Go to the *Software AG_directory* \Designer\eclipse directory and open the `eclipse.ini` file in a text editor.

2. Add the following property at the end of the file: `-Dosgi.nls.warnings=ignore`.

Note:

You can also add this property to the server configuration file, located here: *Software AG_directory \profiles\server_default\configuration\config.ini*. Make sure you restart the server when you update server configurations.

3. Save the file.

Enabling the OSGi Console

If you are familiar with OSGi, or you want to expand your knowledge about it, you can use the OSGi console while working with Application Platform. The OSGi console can be useful for troubleshooting. It is available in the Terminal view of Designer. However, in order to use the Terminal view in the App Platform perspective, you must first enable the OSGi console.

CAUTION:

The OSGi console uses unsecured telnet. Make sure that the OSGi console is disabled on production systems.

> To enable the OSGi Console

1. Go to the *Software AG_directory \profiles\IS_default\configuration\config.ini* directory and open the `config.ini` file in a text editor.
2. In the `config.ini` file, enter `osgi.console=[port_number]`. Save the file.
Specify an unused value for the port number.
3. If the Integration Server server is running, stop and restart the server to apply the changes.
4. In Designer, configure the Terminal view to connect to the port specified in step 2.
5. Open the Terminal view and press Enter.
An `osgi>` prompt appears.
6. To view the OSGi console help, type `help` in the console.

About the Application Platform Perspective

Designer uses perspectives to organize a set of editors and views in the workbench, which are provided for specific development tasks. Application Platform provides the custom App Platform perspective for developing Application Platform projects. The App Platform perspective contains a collection of default views. Many of the views in the App Platform perspective are core Eclipse components. For information about the Eclipse views, see the Eclipse documentation at <http://help.eclipse.org>.

Note:

You can customize the App Platform perspective by using core Eclipse tooling. To return to the default state of the perspective click the **App Platform** button, and then click **Reset**.

Application Platform Designer Views

The default layout of the App Platform perspective contains the following views:

- **Project Explorer.** Access projects.
- **Package Explorer.** Display the Java element hierarchy of the Java projects in your workbench.
- **Main Code Editor.** Edit selected resources.
- **Outline.** Display an outline of the current resource in the code editor window.

Note:

Not every resource will have content in the Outline view.

- **Properties.** Display properties of the current resource in the active view.

Note:

Not every resource will have content in the Properties view.

- **Servers.** Start or stop the server and to publish or unpublish Application Platform projects.
- **Problems.** Resolve errors, such as compilation errors in project source files.
- **Javadoc.** Display Javadoc source documentation for the selected Java source file in the code editor window.
- **Console.** Display content, written to the system IO streams, `stdout` and `stderr`, or read from the process input, `stdin`.
- **Error Log.** Display messages, written to the Designer's log file, which is located here:
`workspace_directory/.metadata/.log`.
- **Bundle Publisher.** Publish additional bundles to the server, or to unpublish bundles from the server.

For more information, see [“Configuring Bundle Publisher View” on page 76](#).

- **Bundle Manager.** Create or delete wrapper bundles that wrap non-OSGi jars.

For more information, see [“Configuring Bundle Manager View” on page 77](#).

- **Terminal.** Open a telnet connection to the OSGi console of the server profile.

Note:

This view requires additional configuration.

For information about configuring the OSGi console, see [“Enabling the OSGi Console” on page 23](#).

For more information about the different views, see the *Workbench User Guide* in *Software AG Designer Online Help*.

Application Platform Context Menu

Application Platform has its own context menu for executing wizards and utilities. The tools in the menu are divided in the following categories:

- **Core Tools.** These tools are available regardless of the server product used. The **Create Project Bundle** and the **Create Bnd template** menu items are listed here.
- **Product-Specific Tools.** These tools are available for specific server products. For example, the product-specific tools for Integration Server are located in the **IS Tools** submenu.

Creating Project Bundles

You can create OSGi bundles for your Application Platform projects from the **App Platform** context menu. The new bundles are located in an artifacts folder, which resides in the current Designer workspace. For example, if you are creating a bundle for a project named `MyJavaProject`, it will be created in the following location:

```
workspace_directory/.metadata/.plugins/com.softwareag.ide.eclipse.pld.bundle.builder.ui/MyJavaProject/artifacts/.
```

Note:

You can create project bundles as a diagnostic tool without defining a server configuration and publishing your project to a server. It is not required to publish bundles to a server.

» To create a project bundle

1. Go to the Package Explorer view and right-click the required project.
2. Select the **App Platform** context menu and click **Create Project Bundle**.

Creating and Customizing Bundle Tool Templates for Projects

You can create Bundle Tool (Bnd) template files for your Application Platform projects. The Bnd template files are located in the `src/main/resources/OSGI-OPT` directory. Bnd template files are useful when the default manifest file produced during bundle creation requires additional customization.

For information about Bnd templates in Application Platform, see [“Bundle Tool Templates for Projects” on page 34](#).

For information about the syntax and supported options of Bnd templates, see <https://bnd.bndtools.org/>.

Important:

Make sure that all Java package names in your Application Platform web project begin with a lowercase character. If the name of a package begins with an uppercase character, the Asset Build Environment does not move the package to the `WEB-INF\classes` directory when you build your project and the package is not available at runtime.

> To create and customize a Bnd template for a project

1. Go to the Package Explorer view and right-click the required project.
2. Select the **App Platform** context menu and click **Create Bnd template**.
3. Go to the Package Explorer view and double-click the Bnd template file, located here:
`project_name/src/main/resources/OSGI-OPT/bnd.bnd`.

Designer loads the `bnd.bnd` file.

4. Edit the `bnd.bnd` file by adding the required custom values.
5. Save the `bnd.bnd` file.

About Creating Application Platform Projects

Application Platform includes two project wizards in the App Platform perspective. The wizards create projects that meet the requirements for publishing projects to the server.

- **Web Project.** Create servlet-based projects.
- **Java Project.** Create all other projects.

You can use other project wizards for developing Application Platform projects. To do this, you must first select additional Application Platform project facets.

For more information about using other project wizards for developing Application Platform projects, see [“Configuring a Designer Project for Application Platform ” on page 35](#).

Selecting Project Facets

Application Platform project wizards utilize project facets to capture additional configuration required for publishing projects to the server. In the Project Facets wizard page you can view a list of all project facets registered in Designer. When you select a project facet, it performs validation for its specific requirements. For example, you must first select the Application Platform Core and Java facets before selecting any other Application Platform facets.

Note:

Some project facets have their own wizard pages that supports additional configuration. The order and number of wizard pages displayed in Designer will vary based on the selected project facets.

➤ To select Application Platform project facets

1. In the Project Explorer view of the App Platform perspective, right-click your project and then click **Properties**.
2. In the Properties dialog box click **Project Facets**.

Designer lists the available facets for the selected project, together with the facet version numbers.

3. Select the check boxes next to the facets you want to add to your project.
4. Click **Apply**, and then click **OK**.

Selecting an Application Platform Runtime Environment

Before you publish your project to the server, you must select a server runtime environment.

➤ To select the server runtime

1. In Designer, go to the Project Explorer or Package Explorer view and right-click your project.
2. Click **Properties**.
3. In the Properties dialog box click **Targeted Runtimes**.
4. Select the check box next to the required runtime environment.

All runtime environments that you have created for Application Platform are listed here.

For detailed steps for creating a runtime environment, see [“Adding a Server Runtime Environment” on page 18](#).

5. Click **Apply**, and then click **OK**.

Creating Java Projects

The App Platform perspective has its own Java Project wizard, which is different from the Java Project wizard of the Java perspective. You can use the Java Project wizard for creating application components that do not require servlet support.

➤ To create a Java project in Application Platform

1. In Designer, go to **File** menu and select **New**.

Designer displays a context menu with all available wizards. The upper section lists the Application Platform wizards.

2. In the upper section of the context menu, click **Java Project**.
3. Specify the settings on the App Platform Core Service Template page.

The following table describes what you have to specify for each setting.

For this setting	Specify
Project name	The name of your Java project.
Use default location	Clear this check box if you want to specify a custom location for your project. If you keep this check box selected, your project will be stored in the default location. Default: check box is selected

4. Click **Next**.
5. On the Project Facets page, select the Application Platform project and core Java project facets required for your Java project.

The Java facet is the core project facet, which is required for Java projects.

6. Go to the Project Facet list, expand **SoftwareAG Application Platform**, and select **Integration Server Extensions**.
7. Click **Next**.
8. Optionally, on the Java page, modify the project's folder structure and default output folder.

Application Platform requires the Java source directory to follow the Maven 2 convention. The Designer project wizard automatically updates the default directory of the Java facet from `src` to `src/main/java`.

9. If you have selected **Integration Server Extensions** in step 6, click **Next** and continue to the next step. Otherwise, click **Finish**.
10. On the App Platform IS Facet page, enter a source path in **Generated Source Path**.

This source path will be added to the Application Platform project's classpath.

11. If the specified source path does not exist on the file system, select the **Include Generated Source Path** check box.

Important:

If you are deploying the project to a project environment using Asset Build Environment and Deployer, verify that the source path is `src/main/java`.

12. Click **Finish**.

Folder Structure of Java Projects

When you create a Java project, Application Platform creates a folder structure that contains the following folder types:

- **Source Folders.** The source folders contain the Java source files and unit test source code. The location path of the source folders must follow the Maven convention to be compatible with the Application Platform. The required location path structure for the source folders is `src/main/java`.

Note:

If you add your unit tests in the source folders, they will be included in the project bundle when you publish your project from Designer.

- **Config Folder.** The `src/main/config` directory contains the property files with configuration data to be passed to the server. When you publish your project bundle to the server, the files in this directory are extracted from the bundle and installed to a common directory on the server, which contains all configuration files for that server.

For more information about configuring projects dynamically, see [“Using the Project Dynamic Configuration” on page 97](#).

- **Resource Folder.** The resource folder contains all non-Java source files. Files and folders that you define in this directory are included in the root directory path of the project bundle.
- **Lib Folder.** The lib folder contains all non-OSGi jar files that you want to include in the classpath of your Application Platform project.

For detailed steps for including non-OSGi jar files in Application Platform projects, see [“Including Non-OSGi Jars in Projects” on page 35](#).

Creating Web Projects

The Web Project wizard enables you to create servlet-based application components. To create web projects, you must configure the Application Platform Web facet.

➤ To create a web project in Application Platform

1. In Designer, go to **File** menu and select **New**.

Designer displays a context menu with all available wizards. The upper section lists the Application Platform wizards.

2. In the upper section of the context menu, click **Web Project**.

3. Enter a name for your web project.
4. On the App Platform Core Web UI Template page, specify the following:
 - a. In the **Project name** field, enter a name for your web project.
 - b. To create the project at the default location, select the **Use default location** check box.
 - c. To create the project at a different location, clear the **Use default location** check box and browse to the location you require.
5. Click **Next**.
6. On the project facets page, expand **SoftwareAG Application Platform** and select **Application Platform Web**.
7. Optionally, select other project facets and click **Next**.
8. Optionally, on the Java page, modify the project's folder structure and default output folder.
9. Click **Next**.
10. On the App Platform Web Facet page, specify web context information in the **Web Context** field.

By default, this field is populated with the name of your project.

Note:

When you build projects with Asset Build Environment in order to deploy them with Deployer, you must define the web context with the `Web-ContextPath: OSGi` manifest header property.

For more information about defining the web context, see [“Configuring Application Platform Projects” on page 79](#).

11. To complete the web project configuration, click **Finish**.

Folder Structure of Web Projects

When you create a web project, Application Platform creates a folder structure for the project. The folder structure contains all folder types, which are created for Java Projects.

For information about the folder types contained in Java Projects, see [“Folder Structure of Java Projects” on page 29](#).

Additionally, a `src/main/webapp` directory is created for web projects. Use this directory for web-related content, such as HTML, JSP, JavaScript, and CSS.

Creating a Dynamic Web Project for Integration Server

This section describes the steps you must execute to create a core Designer Dynamic Web Project and configure it for Application Platform and Integration Server.

➤ To create a Dynamic Web Project for Integration Server

1. In Designer, go to **File** menu and select **New**.

Designer displays a context menu with all available wizards.

2. In the lower section of the context menu, under the Web folder, click **Dynamic Web Project**.
3. Enter a name for your web project.
4. Under Project location, do one of the following:
 - a. To create the project at the default location, select the **Use default location** check box.
 - b. To create the project at a different location, clear the **Use default location** check box and browse to the location you require.
5. Under Target runtime select **Application Platform Integration Server**.
6. Under Configuration select the **App Platform Web Archive (WAR) Preset**.

Important:

If you are deploying the project using Asset Build Environment and Deployer, you must set the generated source directory to `src/main/java`.

7. To complete the web project configuration, click **Finish**.

Classpath Containers

The classpath containers are a collection of libraries, that you can add to the classpath of your project. The following classpath containers are available for Application Platform:

- **Application Platform API Libraries** - contains a Software Development Kit (SDK) with core and server-specific features. By default, Designer automatically adds this classpath container when you create an Application Platform project.

The Application Platform API Libraries container includes the following:

- Application Platform Core - the core annotations related to publishing POJOs as OSGi and Integration Server services, for example `@Service`, `@ServiceReference`, and `@ExposeToIS`.
- Application Platform IS - the Integration Server SDK bundles.

- Third party bundles, provisioned for use with Software AG products.
- **Application Platform Shared Bundles** - contains the shared bundles of Application Platform for a project. You can configure a different set of shared bundles (or libraries) for each project.

Configuring the Application Platform API Libraries Container

Designer automatically adds the Application Platform API Libraries container to the build path of every Application Platform project you create or import. By default, the Application Platform API Libraries container initializes with the Integration Server profile, but contains only the Application Platform Core libraries.

Selecting a profile doesn't require using server-specific libraries in your project. You can add only the Application Platform Core library, or third-party features to the Application Platform API Libraries container.

You can add the Application Platform API Libraries container to any project as a library dependency. However, not all classes provided by this container work for all projects, created in Designer. Some of the SDK functionality, like the service publishing annotation functionality, is only available for Application Platform projects. For more information, see the *Application Platform API Guide*.

> To configure the Application Platform API Libraries container for your project:

1. In the Package Explorer view, right click the Application Platform API Libraries container for your project and then click **Build Path**.
2. Click **Remove from Build Path**. Application Platform removes the default Application Platform API Libraries container.
3. Right-click your project, and then click **Build Path**.
4. Click **Add Libraries....**
5. In the Add Library dialog box, select **Application Platform API Libraries**, and click **Next**.
6. Select the **IS_default** server profile, and click **Next**.
7. On the **SDK Features List** page, select the libraries to add with the container and click **Finish**.

Adding Libraries to the Application Platform API Libraries Container

The default Application Platform API Libraries container has only the Application Platform Core features. Use the following procedure to add more features, or individual libraries in a feature, to the Application Platform API Libraries container that you configure for your project.

> To add libraries to the Application Platform API Libraries container:

1. In the Package Explorer view, right click the Application Platform API Libraries container for your project and then click **Properties**.
2. On the **SDK Features List** page, do one of the following:
 - Select the features to add to the Application Platform API Libraries container for your project.
 - Expand a feature and select individual libraries to add to the Application Platform API Libraries container for your project.
3. Click **OK** to confirm your selection.

Adding the Application Platform Shared Bundles Container

The Application Platform Shared Bundles container has a collection of libraries, which you can add to the classpath of your project. You can use this container for shared bundle dependencies. Each project can define its own classpath container for the Application Platform shared bundles container. The classpath container enables you to specify the location of the shared bundles in your file system. Make sure that you keep only valid OSGi bundles in the shared bundles directory. Any non-OSGi jars that are contained in the shared bundles directory will not be included in the library entry.

For more information about bundle dependencies, see [“About Managing Project Dependencies” on page 69](#).

➤ To add the Application Platform Shared Bundles container

1. In Designer, go to the Package Explorer view, right-click your project, and select **Build Path**.
2. Click **Add Libraries**.
3. In the Add Library dialog box, select **Application Platform Shared Bundles**, and then click **Next**.
4. In the Edit Variable Entry dialog box, specify values in the provided fields.

The following table describes what you have to specify for each field.

In this field	Specify
Name	The name of the classpath variable. Keep the default value. Default: BUILD_EXTERNAL_DIR

Note:

The BUILD_EXTERNAL_DIR variable points to the folder location that contains your project’s external dependencies.

In this field	Specify
Path	<p>The file path to the shared bundles container in your local directory. To specify the path, click Folder, navigate to the directory that contains the bundles, and click OK. All bundles in the selected directory will be added to the classpath of your project.</p> <p>When you specify a path here, you can select the folder structure under that path. Make sure that the selected directories contain valid third-party or external OSGi bundles that are added to the project as library dependencies for compilation purposes. Note that the jars in the selected directories are not included within the project bundle when it is built and deployed to the configured server runtime.</p>

The directories that you configure in this step are added in the project `assetBuild.properties` file in the `component.dependencies.external` property. This value is used by the Asset Build Environment when building assets to resolve the external dependencies used by the project. The equivalent property in the Asset Build Environment that points to the global external build directory is called `build.external.dir` and is configured in the master build properties file of the Asset Build Environment.

For more information about the `assetBuild.properties` file, see “[Application Platform Project Configuration for Asset Build Environment](#)” on page 95.

- In the Edit Variable Entry dialog box, click **OK**.
- Click **Finish**.

The bundles that are located in the selected directories relative to the `BUILD_EXTERNAL_DIR` folder value are listed as library dependencies for the project.

If you add, remove, or change bundles in your local directory, refresh your project and build it to ensure that the classpath of your project is updated.

Important:

If you update the `BUILD_EXTERNAL_DIR` classpath variable to a different location on the file system, you must also update the projects that use this shared classpath container and select the correct folder(s) that contain the libraries to be added to the classpath.

Bundle Tool Templates for Projects

When you publish a project to a server, an OSGi-compliant manifest file is automatically generated for the project in the `src/main/resources/META-INF` directory. This manifest contains default values for the minimal set of required OSGi headers. If you need additional values, you must dynamically customize the manifest file. For this purpose, you must create a Bnd template file for your project and include it in your project’s source control. When you create a Bnd template, the default template is added in the `src/main/resources/OSGI-OPT` directory of your project. You can customize the default Bnd template with a text editor. When your Application Platform project directory contains a Bnd template, the contents of the project manifest file are dynamically updated with the contents

of the template every time you publish the project. If no Bnd template exists for a project, Application Platform uses the default contents of the manifest file when you publish the project.

For information about how to create a Bnd template and add values to it, see [“Creating and Customizing Bundle Tool Templates for Projects” on page 25](#).

Note:

Use different package names for different projects, because Application Platform exports all packages by default. When you are working on more than one project, Application Platform may export the same package and version to different bundles. In such cases you can use split packages to avoid runtime errors.

Including Non-OSGi Jars in Projects

You can include libraries in your project’s `lib` folder. The libraries that you add in this folder will be included in the classpath of your project bundle. In this way you can include non-OSGi jars in your project bundle’s classpath. The jars you add in the `lib` folder will be available only to your project’s classes. To include non-OSGi jars in your project, navigate to your project’s `lib` folder and add the files you require in it.

For example, if you add the `jfind.jar` in the `lib` folder, the generated project bundle will contain the `jfind.jar` and the OSGi Bnd template will contain the following header attribute:

`Bundle-ClassPath: ., lib/jfind.jar`. The OSGi container will include the classes of your jar file in your project bundle’s classpath when you publish the bundle. However, the packages of the jar file will not be exported and will be resolved only by classes inside the bundle.

For more information, see [“About Managing Project Dependencies” on page 69](#).

Configuring a Designer Project for Application Platform

This section describes the steps you must execute, for the most common cases, in order to import a Designer project that was not created with an Application Platform project wizard.

Note:

You can safely change the selected Application Platform project facets for a project. No files are deleted when you uninstall one of the project facets.

Configuring an Application Platform Java Project

This section describes the steps you must execute for a project that was created in Designer using the basic Java Project wizard from the Java perspective.

➤ To configure an Application Platform Java Project

1. In Designer, go to the Package Explorer or Project Explorer view, right-click your project, and then click **Properties**.
2. In the Properties dialog box click **Project Facets**, and then click **Convert to faceted form....**

3. Verify that the Java project facet for the appropriate version is selected.
4. Expand **SoftwareAG Application Platform** and select **Application Platform Core**.
5. Optional. Select **Integration Server Extensions**.
You need this project facet if you are using the IS Service wizard.
6. Optional. Click **Further configuration available** to provide additional configuration.
For information about the additional configuration, see [“Creating Java Projects” on page 27](#).
7. After you are done with the configurations in the Properties dialog box, click **OK**.
8. Verify that the following modifications on your project are successfully executed:
 - a. The source code folder is moved from `src` to `/src/main/java` and it is selected in the project's Build Paths dialog box.
 - b. A `src/main/resources` folder is created and it is selected in the project's `assetBuild.properties` file.
 - c. A `/lib` folder is created.
 - d. A `src/main/config` folder is created.

Configuring an Application Platform Dynamic Web Project

This section covers the steps you must execute for a servlet-based project that was created using the **Dynamic Web Project**.

> To configure an Application Platform Dynamic Web Project

1. Go to the `settings` folder in the project's directory in the workspace of Designer.
2. Open `org.eclipse.wst.common.project.facet.core.xml` in a text editor.
3. Remove the fixed element for the `jst.web` facet and save the file.
4. Go to the Package Explorer or Project Explorer view, right-click your project and click **Refresh**.
5. Right-click your project again and click **Properties**.
6. Click **Project Facets**.
7. Verify that the Java project facet for the appropriate version is selected.

8. Clear the **Dynamic Web Module** check box.
9. Expand **SoftwareAG Application Platform** and select **Application Platform Core** and **Application Platform Web**.
10. Optional. Select **Integration Server Extensions**.
You need this project facet if you are using the IS Service wizard.
11. Optional. Click **Further configuration available...** to provide additional configuration.
12. After you are done with the configurations in the Properties dialog box, click **OK**.
13. Verify that the following modifications on your project are successfully executed:
 - a. The source code folder is moved from `/src` to `/src/main/java` and is selected in the project's Build Paths dialog box.
 - b. A `/lib` folder is created.
 - c. An `src/main/config` folder is created.
 - d. An `src/main/webapp/WEB-INF` folder is created.
14. In Designer, move the project's servlet content, for example JSP, CSS, JavaScript, or HTML, from its current location to `src/main/webapp/`.

For example, if the images of the servlet-based project are located in the `C:\WebContent\images\` directory, you should move the images folder to `C:\src\main\webapp\images\`.

About Developing Web Applications for Integration Server

Application Platform provides the `WmAppPlat` package, which enables you to develop new web and service applications and create IS services in your projects. You can then deploy these Application Platform applications to Integration Server. The `WmAppPlat` package supports the `webMethods` tag library for JSP, which was formerly supported by the `WmTomcat` package.

For more information about invoking IS services and the `webMethods` tag library for JSP, see *Getting Started with the Application Platform API*.

The `WmAppPlat` package is installed together with Application Platform Server. Before you start developing Application Platform applications for Integration Server, add the `WmAppPlat` package to the required Integration Server instance and verify that the package is loaded without any errors or warnings. For more information about the `WmAppPlat` package and updating packages in Integration Server, see *webMethods Integration Server Administrator's Guide*.

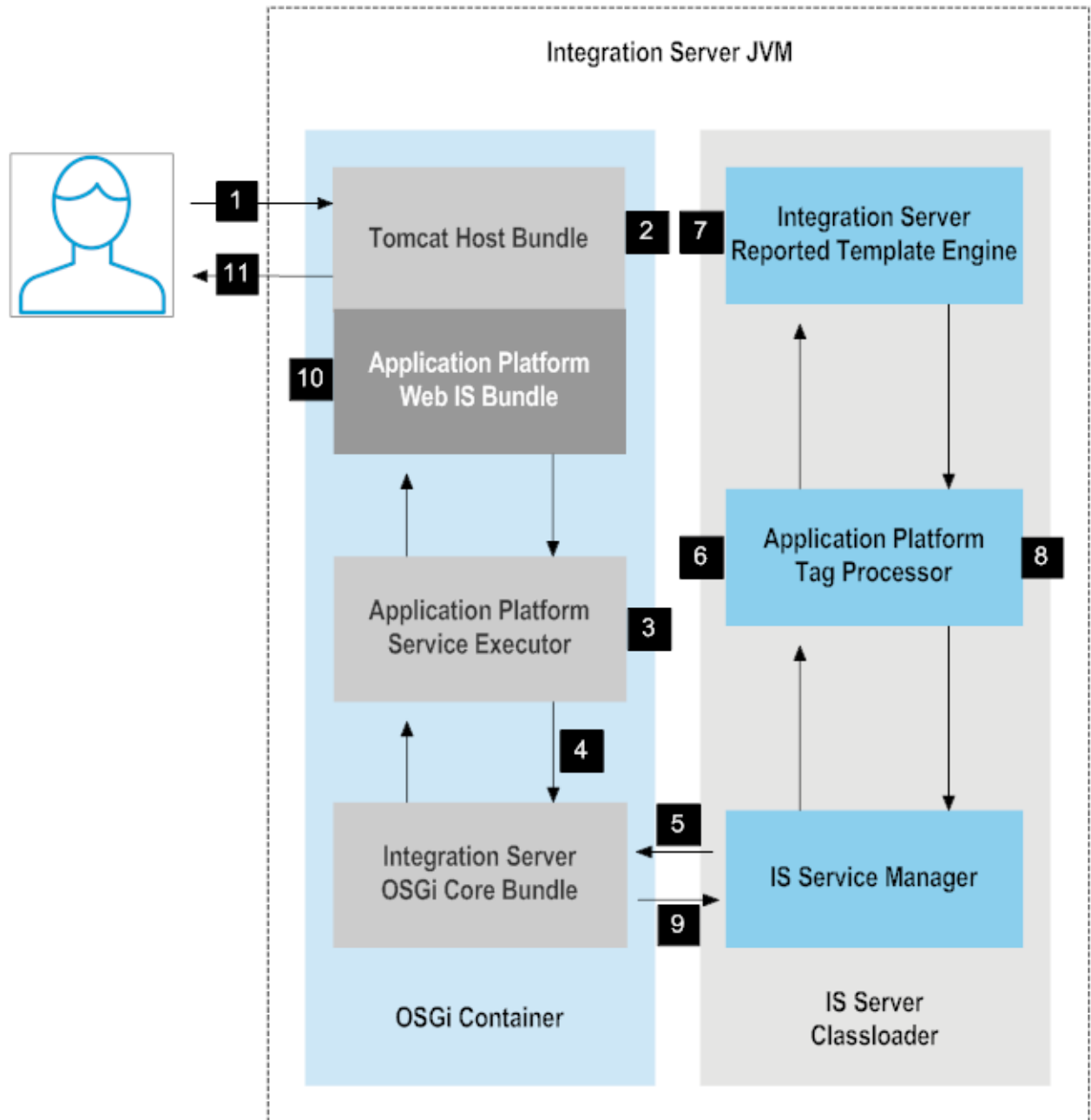
You can migrate existing web applications, developed using the WmTomcat package to Application Platform. For information about migrating web applications to Application Platform, see *Using Software AG Update Manager*.

Processing Web Applications in Application Platform

Application Platform supports Tomcat requests through the WmAppPlat package. By default, the WmAppPlat package is installed with Application Platform. The WmAppPlat package uses Software AG Runtime. Application Platform processes Tomcat requests as follows:

1. The user sends a request to an HTTP port listener hosted by Tomcat that runs in an OSGi container on Software AG Runtime.
2. The HTTP port listener receives and processes the user request.
3. If the JSP page contains a `webm` custom tag, the Application Platform fragment bundle processes the request and parses and provides the tag attributes to another Application Platform bundle, responsible for processing IS service invocations.
4. The Application Platform bundle sends the parsed tag attributes to the IS core bundle.
5. The IS core bundle creates a bridge between the OSGi bundles and the Integration Server classloader and sends the IS service invocations to the Integration Server service manager.
6. The WmAppPlat package processes all `webm` tag declarations through a tag processor using a collection of IS services.
7. The tag processor directly invokes the service requests to the Integration Server reporting engine.
8. The Integration Server reporting engine returns the results of the service requests back to the tag processor.
9. The HTTP response is sent back through the Integration Server classloader and the OSGi container layers to the Application Platform `web.is` fragment bundle.
10. The Application Platform `web.is` fragment bundle sends the response content to the Tomcat catalina response stream.
11. The Tomcat connector returns the response to the user.

The following diagram visualizes this workflow:



Administering Web Applications

In Application Platform you can administer web applications in the following ways:

Production Deployment

When you deploy your web applications to a production environment, use the Asset Build Environment and Deployer. For more information, see [“About Deploying Projects” on page 94](#).

Tomcat Manager Application

If you have Apache Tomcat and you want to use the Tomcat Manager Application in Application Platform, you must install it under *Software AG_directory* /profiles/IS_instance/workspace/webapps/directory. For more information, see the Apache Tomcat documentation.

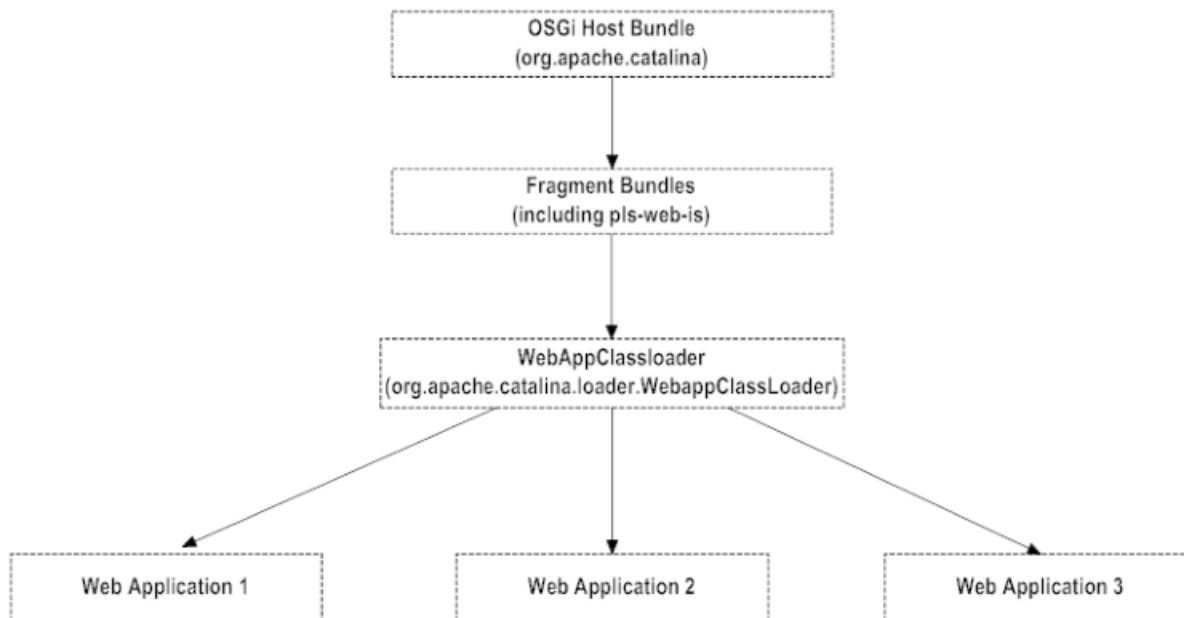
After you install the Tomcat Manager Application, Software AG recommends that you remove the security constraints that the application provides from the `web.xml` configuration file and replace them with the security constraints that the Application Platform `META-INF/context.xml` configuration file provides.

JMX

Application Platform supports Java Management Extensions (JMX). It is registered under the Catalina root domain Mbean. You can view the JMX attributes of your WAR projects deployed to Integration Server in the Properties view of Designer.

Class Loading in WmAppPlat

The following diagram displays how classes are loaded for web applications, running on WmAppPlat. From top to bottom, the OSGi host bundle (`org.apache.catalina`) is extended by a set of fragment bundles, including the `pls-web-is` bundle, which are parent to the Tomcat `WebAppClassLoader`, which is parent to your web applications.



The following table describes each classloader:

Classloader	Description
OSGi host bundle	This classloader is created for the <code>org.apache.catalina</code> bundle.

ClassLoader	Description
fragment bundles	A composite set of fragment bundles extends the complete classpath of the OSGi host bundle. The <code>pl-s-web-is</code> bundle is part of this bundle set.
org.apache.catalina.loader. WebappClassLoader	This classloader is parent to the deployed web applications.
<i>webappN</i>	Each web application has its own classloader, which resolves resources in its context.

Fragment bundles have the following declaration in the `manifest.mf` file: `Fragment-Host: org.apache.catalina`. You can share third-party libraries across your web applications by creating a wrapper bundle with this declaration. This will ensure that any code that is executed in the OSGi host bundle or any fragment bundle will have access to the package imports and exports.

For information about the `manifest.mf` file, see [“Manifests and Bnd Templates for Software AG Common Platform” on page 63](#).

For information about creating wrapper bundles, see [“Creating Wrapper Bundles” on page 73](#).

Developing Web Archive (WAR) Projects

You can use webMethods Application Platform to develop traditional Java servlet Web Archive (WAR) projects. You can deploy the WAR applications you build with Application Platform to Apache Tomcat servlet container.

For more information about developing traditional WAR applications, see Oracle documentation.

Developing Web Application Bundle (WAB) Projects

You can use webMethods Application Platform to develop Web Application Bundle (WAB) projects. While the WAB projects you develop are servlet applications, they are in essence OSGi bundle jars.

About Adding Single Sign-on Authentication in Application Platform Projects

Application Platform enables you to set up security configurations to your servlet-based web applications. Depending on your project and requirements, you can use the available security filter, class, or annotation in your Application Platform projects. The class and the annotation are available in the Application Platform API Libraries classpath container. For more information about the Application Platform API Libraries container, see [“Configuring the Application Platform API Libraries Container” on page 32](#).

You can enable SSO authentication and authorization in your Application Platform projects by adding SSO:

- To the web application layer of an Application Platform web project by using the standard Java EE approach of configuring a security filter in the web.xml file, which is the deployment descriptor of your project. With the security filter you can add a SSO functionality to your web applications.
- To the OSGi service layer of an Application Platform Java or web project. You can implement SSO to OSGi services that are:
 - Exposed as Integration Server (IS) services and invoked through HTTP(S) calls.
 - Exposed as POJO OSGi services.
 - Invoked from the Application Platform web layer. This layer consists of web applications that run on Tomcat and are deployed to Integration Server via HTTP(S).

For detailed information about the security filter, class, and annotation you can use for adding SSO, see *Getting Started with the webMethods Application Platform API*.

Securing Web Application Bundle (WAB) Projects

The following procedure describes the steps you must execute to secure the web application layer of WAB projects by using the web security filter. This function is available only for Application Platform web projects.

➤ To secure the web application layer of your Application Platform WAB project

1. In Designer, go to the Project Explorer or Package Explorer and right-click your project.
2. Double-click the web.xml file, located here: `project_name/WebContent/WEB-INF/web.xml`.
3. Edit the web.xml file by adding the following entry:
`com.softwareag.applatform.pls.security.filter.AppPlatformSecurityFilter` Configure the filter by specifying one or more of its parameters.

For information about the security filter and its parameters, see *Getting Started with the Application Platform API*.

4. Optional. If the project does not have a Bnd template:
 - a. Go to the Project Explorer or Package Explorer and select **App Platform**.
 - b. Click **Create Bnd template**.
5. Optional. If your project is deployed to Integration Server, edit the `Require-Bundle` key from the `bnd.bnd` file by adding the following entry:

```
com.softwareag.applatform.pls.security, com.softwareag.applatform.  
pls.security.is
```

The `com.softwareag.applatform.pls.security` entry is a security bundle, which ensures that the SSO support can be enabled for this POJO service and resolves the `AppPlatformSecurityFilter` class.

6. Optional. If you are configuring an Application Platform project created in version 9.9 or earlier, redeploy your project to a runtime, configured in a later release of Application Platform than version 9.9.

Securing Web Archive (WAR) Projects

The following procedure describes the steps you must execute to secure the web application layer of WAR projects by using security constraints. This function is available only for Application Platform web projects.

Note:

You can also secure WAR projects by using the annotations, provided by Application Platform. For information about using annotations, see [“Securing the OSGi Service Layer” on page 44](#).

➤ To secure the web application layer of your Application Platform WAR project

1. In Designer, go to the Project Explorer or Package Explorer and right-click your project.
2. Double-click the `web.xml` file, located here: `project_name/WebContent/WEB-INF/web.xml`.
3. Edit the `web.xml` file by adding the required security constraints.

For information about the security constraints, see *Getting Started with the Application Platform API*.

4. Optional. If the project does not have a Bnd template:
 - a. Go to the Project Explorer or Package Explorer and select **App Platform**.
 - b. Click **Create Bnd template**.
5. Optional. If your project is deployed to Integration Server, edit the `Require-Bundle` key from the `bnd.bnd` file by adding the following entry:

```
com.softwareag.applatform.pls.security, com.softwareag.applatform.pls.security.is
```

The `com.softwareag.applatform.pls.security` entry is a security bundle, which ensures that the SSO support can be enabled for this POJO service and resolves the `AppPlatformSecurityFilter` class.

6. Optional. If you are configuring an Application Platform project created in version 9.9 or earlier, redeploy your project to a runtime, configured in a later release of Application Platform than version 9.9.

Securing the OSGi Service Layer

The following procedure describes the steps you must execute in order to secure the OSGi service layer of your Application Platform Java or web project.

➤ To Secure the OSGi Service Layer of your Application Platform project

1. In Designer, go to the Project Explorer or Package Explorer and right-click your project.
2. Create the required Java class to be published as an OSGi service.

For information about creating classes, see the *Software AG Designer Online Help*.

3. Add the `@Service` annotation to the class, created in step 2.
4. Add the `@Secure` annotation to the class, created in step 2.
5. Optional. To enable declarative security, add the `@AclAllowed` annotation at the class or method level to define an Access Control List (ACL) parameter.
6. Optional. To enable dynamic runtime security, associate the class, created in step 2 with the `SecurityContext` class.

For information about the `SecurityContext` class, see *Getting Started with the webMethods Application Platform API*.

7. Optional. If the project does not have a Bnd template:
 - a. Go to the Project Explorer or Package Explorer and select **App Platform**.
 - b. Click **Create Bnd template**.
8. Double-click the Bnd template file, located here:
`project_name/src/main/resources/OSGI-OPT/bnd.bnd`.

9. Edit the `bnd.bnd` file by adding the following entry:

```
Require-Bundle:com.softwareag.applatform.pls.security
```

Note:

This entry is a security bundle, which ensures that the SSO support can be enabled for this POJO service.

Important:

If your Application Platform project was created in version 9.8 and you are using a custom MANIFEST.MF file instead of a `bnd.bnd` file, you must add the entry to the MANIFEST.MF file.

10. Optional. If you are configuring an Application Platform project created in version 9.9 or before, redeploy your project to a runtime, configured in a later release of Application Platform than version 9.9.

About the Application Platform Integration Test Framework

Application Platform enables you to write regular JUnit tests for your projects and execute them in Designer. When you execute a test, Designer provides you with immediate feedback for the test run.

The Application Platform integration test framework uses the same development-time deployment mechanism to publish the test project bundle to the server and to execute the test class and its methods on the server. Application Platform publishes and executes the tests through a JMX call on a JMX MBean service on the configured runtime. The core project code and the JUnit tests run in the same OSGi environment and access the same libraries and Application Programming Interfaces (API). The JUnit test code can use the deployed project POJO instances and other OSGi services that are available in the container, without mocking or stubbing any interactions.

Note:

If required, your test can still mock certain method calls by using the available Mockito bundle in the runtime.

You can develop JUnit tests by using the custom JUnit classes, provided by Application Platform.

For information about developing JUnit tests for Application Platform projects, see the *Getting Started with the webMethods Application Platform API*.

Creating a JUnit Test for an Application Platform Project

Before you start developing JUnit tests for an Application Platform project, verify the following conditions:

1. The Application Platform project you are testing exists.
2. Integration Server is configured for this project.
3. The project is successfully published to the configured server.

When you develop JUnit tests for your Application Platform projects, you must use the dedicated classes, available in the Application Platform API Libraries container.

For information about the Application Platform API Libraries container, see [“Configuring the Application Platform API Libraries Container” on page 32](#).

For information about the classes, provided for the Application Platform integration test framework, see the *Getting Started with the webMethods Application Platform API*.

➤ To create a JUnit test for an Application Platform project

1. In Designer, go to the Project Explorer or Package Explorer view and right-click your project.

2. Create a source folder for your JUnit tests.

For example, `src/test/java`.

For information about creating source folders, see the *Software AG Designer Online Help*.

3. Optional. Create a source folder for your JUnit test resources.

For example, `src/test/resources`.

4. Associate your JUnit test with the Application Platform integration test framework. Depending on the type of your JUnit test, do one of the following:

- If your JUnit test does not extend from another class, create a new package and add a new test class, which extends from the `com.softwareag.applatform.sdk.test.framework.AppPlatformIntegrationTest` class. For information about creating packages and classes, see the *Software AG Designer Online Help*.
- If your JUnit test already extends from another class and cannot use the `AppPlatformIntegrationTest` class as its superclass, add the class type annotation `@RunWith` from the `com.softwareag.applatform.sdk.test.framework.IntegrationTestRunner` class. This will ensure that the custom Application Platform JUnit runner is responsible for running your test.

5. Add the `@TestBundle` annotation to the test class, created in step 4, and do the following:

- a. Specify the bundle symbolicName property.
- b. If you have specified a version for your project, different from `1.0.0`, update the value of the version property with the correct project version.

6. Add one or more JUnit test methods and annotate them with the `@Test` method.

7. Optional. Specify the details of the configured server by using the `@RunOnServer` annotation.

Executing a JUnit Test for an Application Platform Project

You can execute your JUnit tests in Designer. When you execute JUnit tests, Designer publishes them to the server, together with the core Application Platform project code.

> To execute JUnit tests for an Application Platform project

1. In Designer, go to the Project Explorer or Package Explorer view and expand the project, where your JUnit test is located.
2. Locate and select your JUnit test.
3. Go to **Run** menu and select **Run As**.

4. Click **JUnit Test**.

Designer reports the success and failure messages from the JUnit test in the JUnit view and in the Console view.

Managing Servers

This section describes tasks for configuring Application Platform servers. Server management in Application Platform is based on the Eclipse Server Tools Project. For information about the server tools, see the *Web Tools Platform Guide in Designer's Help Contents*.

Configuring Integration Server for Application Platform Projects

Use the following procedure to configure an Integration Server for your Application Platform projects.

Important:

Always stop the server before changing its configurations. Otherwise, unpredictable results may occur.

➤ To configure an Integration Server for an Application Platform project

1. In Designer, go to the Servers view and double-click the Integration Server you want to configure.
2. In the General Information section, specify values in the provided fields.

The following table describes what you have to specify for each field.

In this field	Specify
Server name	The name of the Integration Server, to which you publish projects. Default: <code>webMethods Integration Server at host_name</code> .
Host name	The host name or address of the Integration Server, to which you publish projects. If you have created a runtime with a local Integration Server, keep the value of this field to <code>localhost</code> . Default: <code>localhost</code> .

3. Select a **Runtime Environment**.

Note:

The runtime environment includes an absolute directory path to the Application Platform installation, together with the relative paths to the platform bundles that represent the server libraries available in your project's classpath. Designer uses this absolute path to locate the server profile when you attempt to start or stop the server.

- To edit the settings used when Designer executes operating system scripts that start or stop the server, click the **Open Launch Configuration** link.

For information about the launch configuration properties, see [“Configuring Launch Configuration Settings for Integration Server”](#) on page 50.

- After you complete the launch configuration, click **OK**.
- In the Publishing section, select one of the provided publishing settings.

The following table describes the options you can select.

Select this option	To
Never publish automatically	Default. If you select this option, you must publish your project to the server manually.
Automatically publish when resources change	<p>If you select this option, your projects will be published automatically with a predefined time interval, in seconds, every time you update a project resource. You can configure the time interval of this setting.</p> <p>Default: 15</p> <p>Note: This option requires a lot of resources.</p>
Automatically publish when resources change	<p>If you select this option, your projects will be published automatically after any project build event (for example clean, full, or incremental project build).</p> <p>Note: This option requires a lot of resources.</p>

- In the Server Properties section, specify values in the provided fields.

The following table describes what you have to specify for each setting.

For this setting	Specify
Instance name for Integration Server	<p>This field matches the instance name of Integration Server.</p> <p>Default: default</p>
Server Port	<p>The HTTP port for the configured Integration Server. This port is used to verify the server startup sequence. Based on the port number you configure here, Designer uses the corresponding default user credentials to connect to Integration Server.</p> <p>Important:</p>

For this setting	Specify
	<p>If you change the user credentials in Integration Server, you must also update them in Designer.</p> <p>For information about editing the credentials used by Designer, see “Editing the Credentials Used for Connecting to Integration Server” on page 56.</p> <p>Default: 5555</p>
Server Debug Port	<p>The Java Platform Debugger Architecture (JPDA) debugger port configured for the JVM that Integration Server uses. The port value is sent to the startup scripts when Integration Server is started by Designer. During the server startup sequence if Designer cannot connect to the configured debugger port, the server still starts. However, any breakpoints will be ignored. For more information about JPDA, see the Oracle documentation.</p> <p>Default: 9191</p>
Server JMX RMI Port	<p>The Java Management Extensions Remote Method Invocation port used to execute a service for publishing bundles to the OSGi container. This port number is configured in Integration Server in a property file, located in the <i>Software AG_directory \ profiles \ Instance_Name_for_Integration_Server \ configuration \ com.softwareag.platform.config.propsloader</i> directory.</p> <p>For more information about the server configuration, see “Software AG Servers” on page 14. For more information about JMX, see the Oracle documentation.</p> <p>Note: If the port number is in use while installing Application Platform, the port number may change in the server configuration. If you are uncertain of the server state, use an operating system utility to see if the JMX port is in LISTEN mode.</p> <p>Note: The default value matches JMX RMI port value, configured in Integration Server.</p> <p>Default: 8075</p>
Server Connection Mode	<p>Select an option for synchronizing the Servers view with the state of the external servers when you start Designer, or when a server is stopped or started outside of Designer. This list box has the following options:</p> <ul style="list-style-type: none"> ■ Debug - Default. If you select Debug, Designer will automatically start debugging the servers in the Servers view. For example, if you

For this setting	Specify
------------------	---------

restart Designer, the server instance in the Servers view will automatically start debugging.

- **No Action** - If you select **No Action**, Designer will not synchronize the state of the Servers view with the server. If Designer is started and a server is running, its Servers view will indicate the server is stopped. In this case you must execute the **Start** or **Debug** action in the Servers view. Also, if the server status changes while Designer is still running, the change will not be indicated in Designer.
- **Run** - If you select **Run**, Designer will automatically set the server status to started and you will not be able to debug applications remotely, while Designer is connected to the server.

Note:

Since the server states are synchronized with a polling mechanism, there may be a short delay when the Servers view is updated.

8. In the Timeouts section, specify values in the following fields:

The following table describes what you have to specify for each field.

In this field	Specify
Start	<p>Configure how long Designer should wait for the server to start before assuming failure. If the timeout is exceeded, you will see an error message in Designer.</p> <p>For more information, see “Server Start Action” on page 59.</p> <p>The default Start value is 300 seconds.</p>
Stop	<p>Configure how long Designer should wait for the server to stop before assuming failure. If the timeout is exceeded, you will see an error message in Designer.</p> <p>For more information, see “Server Stop Action” on page 60.</p> <p>The default Stop value is 60 seconds.</p>

Configuring Launch Configuration Settings for Integration Server

Use the following procedure to configure the launch configuration settings for Integration Server.

- **To configure the launch configuration settings for Integration Server**

1. In Designer, go to the Servers view and double-click the Integration Server you want to configure.
2. In the General Information section, click **Open Launch Configuration**.
3. On the **Server** tab of the Edit Configuration dialog box, select the server to configure from the **Server** list box.
4. Optionally, click the **Arguments** tab and define the arguments to be passed to the application and to the virtual machine, if any.

For information about the Integration Server configurations on the Arguments tab, see [“Configuring Integration Server Arguments” on page 51](#).

5. Optionally, click the **Source** tab and define the location of source files that Designer uses to display the source when debugging your Java applications.

For information about the Integration Server configurations on the Source tab, see [“Configuring the Integration Server Source” on page 52](#).

6. Optionally, click the **Environment** tab and define the environment variables to use when running or debugging your Java applications.

For information about the Integration Server configurations on the Environment tab, see [“Configuring Integration Server Environment” on page 53](#).

7. Optionally, click the **Common** tab and define general information about the launch configuration.

For information about the Integration Server configurations on the Common tab, see [“Configuring Common Integration Server Settings” on page 54](#).

8. Click **Apply**, and then click **OK**.

Configuring Integration Server Arguments

On the Arguments tab of the launch configuration properties for Integration Server, you can configure the Program and VM arguments. The Program arguments are processed by the Equinox OSGi Framework. The VM arguments modify the settings of the Java Virtual Machine (JVM).

➤ To configure the Integration Server arguments

1. In Designer, go to the Servers view and double-click the Integration Server you want to configure.
2. In the General Information section, click **Open Launch Configuration**.
3. In the Edit Configuration dialog box click the **Arguments** tab.

4. In the **Program arguments** field specify program arguments for the Equinox OSGi Framework.

Important:

Do not modify or delete the existing program arguments!

Tip:

Click the **Variables** button to select a variable from the list, or to define your own variables.

5. In the **VM arguments** field specify VM arguments for the JVM.

Important:

Do not modify or delete the existing VM arguments!

Tip:

Click the **Variables** button to select a variable from the list, or to define your own variables.

6. Select the working directory that Integration Server uses for the launched process by clicking one of the provided options.

The following table describes the options you can select from.

Select this option	To
Default	Use the root directory of the Designer installation: <i>Software AG_directory \Designer\ eclipse.</i>
Other	Change the default directory. You can set as a working directory any directory to which you have write privileges. To set a working directory, click one of the following buttons: <ul style="list-style-type: none"> ■ Workspace - select a Designer workspace as a working directory. ■ File System - select a working directory from your file system. ■ Variables - set a variable for the working directory. For instructions about how to set a variable, see the tip in step 5. <p>Default: <i>Software AG_directory \profiles\IS_default\bin</i></p>

7. Click **Apply**, and then click **OK**.

Configuring the Integration Server Source

On the Source tab of the launch configuration properties for Integration Server, you can define the location of source files used to display the source when debugging a Java application. By default, the location is derived from the build path of the associated project.

> To define a new source lookup path

1. In Designer, go to the Servers view and double-click the Integration Server you want to configure.
2. In the General Information section, click **Open Launch Configuration**.
3. In the Edit Configuration dialog box, click the **Source** tab.
4. To add a new source lookup path, click **Add**.
5. In the Add Source dialog box, select the required source files by using one of the provided options.

The following table describes the options you can select from.

Select this option	To add
Archive	A jar or zip in the workspace containing source files.
External Archive	A jar or zip in the local file system containing source files.
File System Directory	A directory in the local file system.
Java Classpath Variable	A workspace folder, a local directory, or an archive referenced by a variable path.
Java Library	A collection of binary archives with the source attached.
Java Project	Source folders in a Java Project.
Workspace Folder	A folder in the workspace.

6. Optionally, select the **Search for duplicate source files on the path** check box to search the source lookup path and include duplicate entries. By default, this check box is cleared.
7. Click **Apply**, and then click **OK**.

Configuring Integration Server Environment

You can configure environment variables on the Environment tab of the launch configuration properties for Integration Server. Designer uses the environment variable values when it runs an application. By default, the environment is inherited from the Designer runtime.

1. In the General Information section, click **Open Launch Configuration**.
2. In the Edit Configuration dialog box, click the **Environment** tab.
3. Click **New** to define a new environment variable.

Important:

Do not modify or delete the existing environment variables!

Tip:

Click the **Variables** button in the New Environment Variable dialog box to select a variable from the list, or to define your own variables.

4. Click **Select** to select one or more native environment variables from a list and add them to your Integration Server launch configuration.
5. Select one of the provided options for the launch configuration environment.

The following table describes the options you can select from.

Select this option	To
Append environment to native environment	Default. Append to the native environment. Designer seeds the launched environment with the native environment, after which the variables configured in the Environment tab replace or augment the set of environment variables.
Replace native environment with specified environment	Replace the native environment. Designer creates the launched environment only from the variables configured on the Environment tab.

6. Click **Apply**, and then click **OK**.

Configuring Common Integration Server Settings

On the Common tab of the launch configuration properties for Integration Server, you can define general settings of the launch configuration.

➤ To define common launch configuration settings for Integration Server

1. In Designer, go to the Servers view and double-click the Integration Server you want to configure.
2. In the General Information section, click **Open Launch Configuration**.
3. In the Edit Configuration dialog box, click the **Common** tab.
4. Set the physical location where the launch will be saved by selecting one of the provided options.

The following table describes the options you can select from.

Select this option	To
Local file	Default. Save the launch file in the local workspace metadata.
Shared file	Move the launch file to a custom location in the workspace. Use to share the launch configuration using a version control system.

- In the **Display in favorites menu** field, select one or more menus where you want your launch configuration to appear.
- Set the encoding you want to use for the launch configuration by selecting one of the provided options.

The following table describes the options you can select from.

Select this option	To
Default - inherited (Cp1252)	Default. Use the default encoding.
Other	To select from the supported encoding standards: <ul style="list-style-type: none"> ■ ISO-8859-1 - Default. ■ US-ASCII ■ UTF-16 ■ UTF-16BE ■ UTF-16LE ■ UTF-8

- Define where to provide input and output data by using the provided check boxes.

The following table describes the check boxes you can select.

Select this check box	To
Allocate console (necessary for input)	Select the check box to allocate a separate console in Designer. Clear the check box if you do not require a separate console for the input data. Default: check box is selected.
Input File	Select the check box to specify an input file, in which you can configure the launch configuration. Clear the check box if you do not require an input file to configure the launch configuration.

Select this check box	To
	Default: check box is cleared.
Output File	Select the check box to specify an output file, in which you can configure the launch configuration. Clear the check box if you do not require an output file to configure the launch configuration. Default: check box is cleared.
Append	Select the check box to append newly added launch configuration data to the output file. Clear the check box to have new data that you add to the output file override the existing configurations. Default: check box is cleared.
	Note: You can select this check box only when Output File is selected.

8. Use the **Launch in background** check box to set background launching.

If this check box is selected, Designer launches the configuration in the background, with a separate job. If this check box is cleared, you will not be able to use Designer until the launch operation is complete. Default: check box is selected.

9. Click **Apply**, and then click **OK**.

Editing the Credentials Used for Connecting to Integration Server

If you configure an Integration Server connection in Designer, Designer stores the credentials that are used for connecting to the server instance. However, if you change those credentials in Integration Server, you must also update them in Designer.

➤ To edit the credentials used by Designer for connecting to Integration Server

1. In Designer, go to **Window** menu, and then click **Preferences**.
2. In the Preferences dialog box, expand **Software AG** and select **Integration Servers**.
3. Select the **Integration Server** instance to update based on the **Port** number.
4. Click **Edit**.
5. In the Edit Integration Server dialog box, update the provided fields, as required.

The following table describes what you have to specify for each field.

In this field	Specify
Name	The name of the selected Integration Server instance. Default: Default
Host	The host name or address of the selected Integration Server instance. Default: localhost
Port	The HTTP port for the selected Integration Server instance. This port is used to verify the server startup sequence. Default: 5555
User	The user name used to connect to the selected Integration Server instance. Default: Administrator
Password	The password used to connect to the selected Integration Server instance. Default: manage

6. Optional. Set the provided check boxes to define connection settings.

The following table describes the check boxes you can set.

Set this check box	To
Connect immediately	When this check box is selected, Designer connects immediately to the Integration Server instance after you complete the connection configuration. When this check box is cleared, Designer connects to the configured Integration Server instance after you start the connection manually. Default: check box is selected
Connect at startup	When this check box is selected, Designer connects to the configured Integration Server instance on startup. When this check box is cleared, Designer connects to the configured Integration Server instance after you start the connection manually. Default: check box is selected
Secure connection	When this check box is selected, Designer connects to the configured Integration Server instance through HTTPS. When this check box is cleared, Designer connects to the configured Integration Server instance through HTTP. Default: check box is cleared

7. In the Edit Integration Server dialog box, click **OK**.
8. In the Preferences dialog box, click **OK**.

Creating a New Integration Server Instance with the Application Platform Support Package

You can create a new Integration Server instance and install the Application Platform Support package to it by using the Integration Server instance script.

➤ To create a new Integration Server instance with the Application Platform Support package

1. Navigate to the following directory: *Software AG_directory / Integration Server_directory /instances*.
2. Run the `is_instance` script with the `create` command and specify the following additional JVM parameter:

```
-Dpackage.list=WmAppPlat
```





Important:

When you add the WmAppPlat package, which is used for Application Platform, the new Integration Server instance includes a configured instance of Tomcat. This Tomcat instance uses ports 8072 and 8074 as the default HTTP and HTTPS ports, respectively. These default ports conflict with the ports used by Tomcat on the default instance of Integration Server . You must use Command Central to change the default HTTP and HTTPS port numbers for Tomcat on the new instance of Integration Server.

For more information about the `is_instance` script, including details about the `create` command and the remaining additional JVM parameters, see *webMethods Integration Server Administrator's Guide*.

Managing Server Status

The Servers view is an Eclipse component. In Designer it is customized for Software AG servers and it allows you to manage the status of your servers. You can perform the provided server lifecycle operations from the actions toolbar in the Servers view. The following table describes the use of each server lifecycle operation.

Use this action	To
	Start the server
	Stop the server
	Debug the server
	Publish or unpublish projects

Note:

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

➤ **To manage the status of a server**

1. Go to the Servers view.
2. Select a server.
3. To change the status of the selected server:
 - Click one of the available actions in the upper right corner.
 - Right-click the server and click one of the available actions.

Important:

If you have configured a remote Integration Server, you cannot start and stop the server in Designer. You must directly start and stop it through the machine, where the server is installed.

For information about Integration Server, see [“Integration Server Lifecycle Actions” on page 59](#).

Integration Server Lifecycle Actions

This section describes the behavior of Integration Server when different server actions are triggered in Designer. For more detailed information, see the *webMethods Integration Server Administrator's Guide*.

For troubleshooting information, see [“Considerations When Publishing Projects to Servers” on page 107](#).

Server Start Action

When you start the server, a shell script is executed. The script must be blocked when the server is started. The runtime environment for the server includes an environment variable that ensures the script is blocked. If the script is not blocked and the server is started asynchronously, Designer will report an error immediately after you attempt to start the server.

When you click the start icon, Designer changes the server status from **Stopped** to **Starting**. Designer uses a polling mechanism to periodically ping the server. When the server starts and responds to the HTTP request, the server status changes from **Starting** to **Started**.

When you start Integration Server, Designer uses the server connection details defined in **Window > Preferences > Software AG > Integration Servers settings** to connect to the Integration Server. Designer executes a GET request by using basic authentication. If the server returns the expected response code within the configured timeout period, the state of the server is changed from **Starting** to **Started**.

Server Stop Action

When you stop the server, a shell script is executed. If the server fails to stop within the configured timeout period, you can terminate the stop action in Designer.

Note:

Terminating the action does not affect the state of the server but it resets the Servers view in Designer.

When you initiate the stop action, Designer uses a polling mechanism to verify the execution of the server shutdown. After the stop action is complete, the server state changes from **Stopping** to **Stopped**.

Server Debug Action

You can start the server by using the debug action. However, there is a difference between the start and the debug action. When Designer updates the state of the server after executing the debug action, it opens a socket connection to the JPDA port. This allows for debugging the source code, which is not possible when you use the start action.

Note:

If you launch the server outside of Designer, make sure that you launch it in debug mode. Otherwise, Designer will not be able to connect to the server because the JPDA port will be closed. To check the state of the server, use an OS utility and verify that the JPDA port is in LISTEN mode.

Server Restart Action

You can restart the server by using the restart, or the restart-in-debug action. The restart action will execute the stop action, and then the start action. The restart-in-debug action will execute the stop action, and then the debug action.

About Publishing Projects

This section describes the processes that are involved when you build Application Platform projects in Designer and publish them to the server. The major phases of project publishing are the following:

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

Building Projects

You must validate and compile an Application Platform project before you can publish it. During the build phase, Application Platform compiles the project source code and produces additional files, for example metadata files. However, Application Platform does not perform any additional bundle-related activities when building a project.

You can build your project either with one of the standard project builders in Designer, or with one of the custom project builders for Application Platform projects. However, Application Platform projects are build with all of the project builders that are automatically configured when you apply the Application Platform Core project facet.

Building Projects with Designer Project Builders

You can use some of the Designer project builders to build your Application Platform projects.

➤ To build your project with one of the standard build actions in Designer

1. In Designer, go to **Project** menu, and then click **Build Project**.
2. Select one of the build actions supported for Application Platform projects.

The following table describes the actions you can select.

Select this action	To
Clean	Purge transient files in your project.
Incremental Build	Build only the resources you modified after the latest build.
Full Build	Build or rebuild the entire project, regardless of the state of the current build.

For more information about the build actions, see the *Workbench User Guide* in *Software AG Designer Online Help*.

Building Projects with Custom Application Platform Project Builders

You can use custom project builders to build your Application Platform projects. The custom builders execute the Designer clean build, incremental build, and full build actions, and at the same time perform Application Platform tasks in the background.

The custom project builders for Application Platform are installed after you enable the Application Platform Core facet for your project.

➤ To build a project with one of the custom build actions

1. In Designer, go to **Project** menu, and then click **Build Project**.
2. Select one of the custom builders supported for Application Platform projects.

The following table describes the builders you can select.

Select this builder	To
Application Platform Builder	Pass additional context information that is captured while your project is compiled to the Project Publisher.
Application Platform Service Publishing Builder	Create additional files that are necessary to publish your project's services in the OSGi container.

Note:

Do not disable or remove the Application Platform project builders from your project.

Publishing Projects to the Server

Before you can publish your project to the OSGi container of a server, you must add your project to the server from the Servers view.

Before you can add and publish a project to the server, make sure that the project is opened in Designer and that it has at least one Application Platform Core project facet.

> To add and publish projects to a server

1. In Designer, go to the Servers view, right-click the required server, and then click **Add and Remove**.
2. Click the required project in the list of available projects, and then lick **Add**.
3. Optional. Remove one or more of the configured projects. In the list of configured projects, click the project you want to remove, and then click **Remove**.

Important:

If you delete a project that is already published to the server, the published project becomes an orphan and you have to remove it manually from the server.

For information about the manual steps, see [“Manually Uninstall a Bundle from the Server” on page 109](#).

4. Optional. Select the **If server is started, publish changes immediately** check box if you want Designer to immediately publish all configured projects after you click **Finish**.
5. Click **Finish**.
6. To publish the configured projects, go to the Servers view and right-click the server to which you want to publish.
7. Select one of the supported publish commands.

The following table describes the commands you can select.

Select this command	To
Publish	Assemble a project bundle from the project files from the most recent build.
Clean	Fully rebuild all configured projects.

Manifests and Bnd Templates for Software AG Common Platform

Before you can create bundles for your project, your project must have a manifest. If a `manifest.mf` file exists in your project's `src/main/resources` folder, the manifest file serves as a template during bundle creation. Otherwise, Application Platform automatically creates a manifest when you publish your project. However, in some cases you need to customize the default manifest. You can customize manifests dynamically using a Bnd template. The following examples describe cases when custom manifests are required:

- **Indirect Package Imports.** The default manifest is created with a list of package imports after analyzing the project classes' imports and locating external package dependencies found in the bundle. Because 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 class references, and are invoked indirectly. These additional dependencies must be exported by another bundle in the container and the manifest must be customized, so that the additional packages can be imported.
- **Reduced Scope of Package Exports.** By default, all packages defined in the project bundle are exported. If you want to customize the package exports, you must declare an alternative set of exports. You can do this by customizing the default manifest. For example, you can configure Application Platform to only export packages that represent a public API, and keep your implementation packages in the bundle.

For information about customizing manifest dynamically using a Bnd template, see [“Creating and Customizing Bundle Tool Templates for Projects” on page 25](#).

Assembling Project Bundles

After you build your project, you must assemble it into a module. To assemble the project to a module, you must create an OSGi bundle. The following major steps are involved in this process:

- Using the default project manifest or creating a Bnd template for Software AG Common Platform.

You can create a custom Bnd template or you can use the default project manifest, provided by Application Platform.

For information about cases when you need to create a Bnd template, see [“Manifests and Bnd Templates for Software AG Common Platform ” on page 63](#).

- **Creating and staging the jar in an Artifacts directory.**

After Designer compiles your project, it inserts the contents of the project into a bundle jar and copies the contents to an Artifacts directory out of the project's workspace. The contents are moved out of the project's workspace because Designer locks the entire workspace while building projects. The artifacts directory is located here: `User_Workspace/.metadata/.plugins/com.softwareag.ide.eclipse.pld.bundle.builder.ui/Project_Name/artifacts/`.

Project bundles you create are also added in the Artifacts directory. After Designer executes a successful clean build action of your project, the associated bundles are removed from the Artifacts directory.

- **Copying the jar to the bundle repository.**

After you publish your project to a server, a JMX service transfers the project bundle from the artifacts directory to the following temporary directory, located in the server's profile: `Software AG_directory \profiles\server_instance\workspace\temp\app-platform\deployer\bundles`. If this transfer is successful, then the project bundle is copied to the following location of the server's repository directory: `Software AG_directory \profiles\server_instance\workspace\app-platform\deployer\bundles`. After the project bundle is successfully copied, the server's publisher service is invoked to install the project bundle into the container's OSGi runtime. When you unpublish the project, Designer removes the project bundle from the server's repository directory and the publisher service removes it from the OSGi runtime.

When you assemble project bundles, the server uses an OSGi service provided by the Common Platform. This service is used for installing and uninstalling bundles from the repository directory. Any errors that occur in this process are added in Designer's error view.

Note:

You cannot add or remove bundles from the server by adding or removing files from the repository directory.

➤ **To assemble the project bundles into a module**

1. Create a manifest or use the default manifest, provided by Application Platform.

For detailed steps, see [“Creating and Customizing Bundle Tool Templates for Projects” on page 25](#).

2. Compile your project.
3. Right-click your project and select **App Platform**.
4. Click **Create Project Bundle**.

About Viewing Dependency Graphs

Application Platform enables you to view a graphic representation of different dependencies. You view dependency graphs in the Visual Navigator view.

The Visual Navigator renders the dependencies as graphs based on different object attributes. Each attribute forms the basis of a *dimension*. Dimensions provide the information necessary to produce a collection of graph nodes and connections between these nodes that illustrate a relationship for a given attribute. A dimension displays node relationships based on one or all of the following relationship types:

- Parent-child composition
- Reference dependencies

A collection of dimensions is called a *universe*. Each Designer perspective can have a single universe registered to it. For example, the App Platform perspective has its own universe, named the App Platform universe. The App Platform universe supports the following dimensions:

- **Project** - Displays an open project from the current Designer workspace. This dimension is useful if you want to see project dependencies.
- **Resources** - Displays composition relationships for a selected resource. The default node depth of this dimension is 1. Increasing the node depth shows more project resources.

For information about node depth, see [“Visual Navigator Node Depth Levels” on page 67](#).

- **Java Packages** - Displays the composition relationships for a selected package. Package graph nodes are rendered in this dimension only if they contain resources, such as files, binary or source classes, or non-class resources. Empty packages are not displayed in this dimension.
- **Classes & Interfaces** - Displays relationships between Java classes and interfaces, as well as ICompilationUnit objects, which construct the Designer JDT (Java Development Tooling) models. ICompilationUnit objects are visible in the Package Explorer and Project Explorer views in Designer. They act as a container for source references.
- **Java Methods** - Displays relationships between the Java methods, used in the selected project.
- **Metadata** - Reserved for internal use.

Opening a Project in the Visual Navigator

You can open your Designer projects in the Visual Navigator view.

➤ To open your Designer projects in the Visual Navigator

1. In Designer, go to the Project Explorer or Package Explorer view.
2. Right-click the required project.
3. Select **Show In** and click **Visual Navigator**.

The Visual Navigator view opens and renders a graph of the selected project.

Using the Visual Navigator

The Visual Navigator provides several functions, which enable you to render the required graph and nodes.


➤ To use the Visual Navigator

1. Open a project in the Visual Navigator view.

For information about how to open a project in the Visual Navigator, see [“Opening a Project in the Visual Navigator” on page 65](#).

2. To view the required graph and nodes, perform one or more of the available actions.

The following table describes the actions you can perform.

To	Do this
Select a dimension	<p>Click the drop-down list box and select one of the available dimensions.</p> <p>The list of available dimensions depends on the currently selected object. The list displays all dimensions that can represent this object.</p> <p>For information about the available dimensions, see “About Viewing Dependency Graphs” on page 64.</p>
Configure the graphic depth	<p>Use the slider, available in the upper part of the Visual Navigator.</p> <p>The positions of the slider represent the available node depth levels, starting from level 0 in the leftmost part.</p> <p>For information about the different node depth levels, see “Visual Navigator Node Depth Levels” on page 67.</p>
Filter the displayed nodes	<p>Enter the required filter text in the text box and click .</p>
Use a context menu command	<p>Right-click a node or anywhere in the graph view and click the required command.</p> <p>For information about the context menu commands, see “Visual Navigator Context Menu Commands” on page 67.</p>
Zoom in or out or reposition the port view	<p>Use the required keyboard shortcut.</p> <p>For information about the available keyboard shortcuts, see “Visual Navigator Keyboard Shortcuts” on page 69.</p>
Open a node in an editor	<p>Double-click the node.</p>

To	Do this
	To open a node in an editor, the Visual Navigator invokes the default editor for the associated resource, defined in Designer. If no default editor is defined for the resource, nothing happens. To view the default editors for different resources, navigate to Window > Preferences > General > Editors > File Associations .

Visual Navigator Node Depth Levels

The Visual Navigator uses node depth levels from 0 to 5 to render the depth of the graphs. You can use these levels to control the scope of information that is displayed on a graph. At level 0, no relationships are displayed between a selected object and its references. Level 5 represents the most complex object relationships.

For example, if you have classes A, B, C, and D, with the following relationship: A > B > C > D, and you want to render graphs at different levels, the following table shows the classes that will be displayed at the different node depth levels you select.

Selected Class	Node Depth Level	Visible Classes
A	0	A
A	1	A, B
A	2	A, B, C
A	3	A, B, C, D
B	0	B
B	1	A, B, C
B	2	A, B, C, D
B	3	A, B, C, D

Visual Navigator Context Menu Commands

The Visual Navigator context menu contains the following submenus and commands:

Show In

The following table describes the commands from the **Show In** submenu.

Command	Description
Terminal	Opens the currently selected node in the Terminal view.

Command	Description
Package Explorer	Opens the currently selected node in the Package Explorer view.
Project Explorer	Opens the currently selected node in the Project Explorer view.
System Explorer	Opens the currently selected node in the System Explorer view.
Properties	Opens the currently selected node in the Properties view.

Other Dimensions

The following table describes the commands from the **Other Dimensions** submenu.

Command	Description
Resources	Renders the Resources dimension, if available.
Java Packages	Renders the Java Packages dimension, if available.
Classes & Interfaces	Renders the Classes & Interfaces dimension, if available.
Java Methods	Renders the Java Methods dimension, if available.
Previous Dimension	Renders the previous dimension, if available.

Navigator Global Actions

The following table describes the commands from the **Navigator Global Actions** submenu.

Command	Description
Refresh view for current dimension	Renders a graph for the currently selected dimension.
Clear view of all graph items	Clears the graph from the Visual Navigator.
Render view on this selected object	Renders a graph for the currently selected node.

App Platform Universe Actions

The following table describes the commands from the **App Platform Universe Actions** submenu.

Command	Description
Show Open Projects	Renders a graph that displays all projects that are currently available in the Designer workspace.

Command	Description
Navigator Metadata	This command is reserved for internal use.

Visual Navigator Keyboard Shortcuts

You can use keyboard shortcuts in the Visual Navigator view. The following table lists the available keyboard shortcuts and the action they perform.

Keyboard Shortcut	Action
Ctrl + -	Zoom out.
Ctrl + =	Zoom in.
Left Arrow	Move the view port to the left.
Up Arrow	Move the view port up.
Down Arrow	Move the view port down.
Right Arrow	Move the view port to the right.
Ctrl + Left Arrow	Reposition the view port to the left with the width of the view port.
Ctrl + Right Arrow	Reposition the view port to the right with the width of the view port.
Ctrl + Down Arrow	Reposition the view port down with the width of the view port.
Ctrl + Up Arrow	Reposition the view port up with the width of the view port.

About Managing Project Dependencies

Applications you develop in Designer may have contents, different from source projects developed in Designer. Application Platform handles such contents as project dependencies. Application Platform supports the following types of dependencies:

- **Dependencies with bundles.** Projects can have dependencies on other bundles. For example, a project can depend on third-party bundles or bundles that are developed by another team.
- **Dependencies with jars that are not bundles.** Projects can have dependencies on plain jar files. Such files can reside in an external location, which under version control or is present in a repository library, such as an artifactory or a Maven repository. You can use plain jar files that are not bundles in the following ways:
 - Include the plain jar files in your project as a local dependency when you publish the project. The common jars are included in your project's `lib` directory and will be duplicated in different projects during publishing.

- Publish the plain jar files as a bundle in the runtime, so that they can be shared with other published projects. The common jars are wrapped as bundles and published once to the runtime. Projects that require one of the common jars refer to the jar's bundle.

For more information about wrapping common jars as bundles, see [“Creating Wrapper Bundles” on page 73](#).

Important:

When you include common jars as bundles, they are not packaged with your project. The common jars are only used for compiling dependencies and for computing the Import-Package OSGi header values of the project manifest or Bnd template while building the project bundle. You must add the referenced bundles in the runtime before you use them. To ensure that bundles can be installed and resolved, set the imports to be required.

Application Platform provides the following views, which you can use for such dependencies:

- **Bundle Publisher View.** Use this view to publish or unpublish a bundle to or from a container.
- **Bundle Manager View.** Use this view to create bundles from non-OSGi jars.

Bundle Publisher View

You can use the Bundle Publisher view to install additional bundles to the server from Designer. You can also uninstall bundles from the server. However, you cannot use the Bundle Publisher view to publish project bundles to the server. You must use the Servers view for publishing.

The Bundle Publisher lists items from the following locations:

- The classpath of the selected project.
- The directory, which is configured in the settings of the Bundle Manager view.

For more information about the settings of the Bundle Manager view, see [“Configuring Bundle Manager View” on page 77](#).

- The server runtime container.

The Bundle Publisher uses different icons in order to distinguish between plain Java jars and OSGi bundles. Depending on the configurations of the Bundle Publisher view, certain items can be excluded from the view, based on the item category.

For more information about the Bundle Publisher, see [“Configuring Bundle Publisher View” on page 76](#).

Publishing and Unpublishing Bundles


You can publish or unpublish bundles by selecting or clearing the bundle check boxes. The Bundle Publisher performs the following operations depending on the check box statuses:

- Installs selected bundles to the server.

- Uninstalls cleared bundles from the server.

You can simultaneously send several publish and unpublish commands to the server.

➤ To publish or unpublish bundles from the server

1. In Designer, go to the Bundle Publisher view.
2. Select the bundles you want to publish to the server.
3. Clear the check boxes for the bundles you want to unpublish from the server.
4. Click .

The selected bundles are published to the server. Published bundles remain selected in the Bundle Publisher view. The bundles with cleared check boxes are unpublished from the server and the newly selected bundles are published to the server.

Bundle Publisher Dependency Graphs

The Bundle Publisher provides you with a dialog, which displays potential warnings and errors that are discovered when validating a collection of bundles. The Bundle Publisher determines the dependencies between the bundles by examining all of the OSGi manifests and Bnd templates in the group of bundles. Based on the examination, the Bundle Publisher forms a dependency graph. When you start the server, all active bundles are also included in the graph, unless you have removed them.

When creating a dependency graph for a group of bundles, the Bundle Publisher validates the attempts for:

- Publishing a bundle that exports the same package and version.
- Publishing a bundle that imports a package that is not exported. Note that you can configure the Bundle Publisher to ignore optional missing imports, like bundles with package imports that contain the following qualifier: `resolution:=optional`.
- Unpublishing a bundle that exports a package imported by another bundle.
- Unpublishing a bundle that is required by one or more published bundles.
- Publishing a bundle that produces a circular dependency.

The following table describes the message types that the Bundle Publisher can display.

Message Type	Description
Information	Status messages that provide information.
Warning	Messages that indicate dependency issues that may prevent bundles from reaching an active state.

Message Type	Description
Error	<p>Messages that indicate invalid bundles.</p> <p>For example, an error message can indicate that there is a corrupt file or a jar with an invalid OSGi manifest.</p>

Examples of Dependency Validation

The following examples describe cases, in which dependency validation is useful:

- Publishing a Bundle with Missing Dependency.** Assume that you have the following bundles in the Bundle Publisher view: *MyProject-A* and *MyProject-B*. Neither of the bundles has been published to the server. *MyProject-B* has a dependency on *MyProject-A* and *MyProject-A* has a dependency on a package (*com.softwareag.demo.c*). The package is not exported by the project, nor by any bundle on the server. This represents a missing dependency. Attempts to publish *MyProject-A* will fail with an error message, because the package import is required.

The following table shows the imports package and the exports package for each bundle.

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

- Removing a Bundle that Provides a Dependency.** Assume that you have the following bundles in the Bundle Publisher view: *MyProject-A* and *MyProject-B*. *MyProject-B* has a dependency on *MyProject-A* and both bundles are published to the server.

The following table shows the imports package and the exports package for each bundle.

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

If you clear *MyProject-A* for unpublishing from the server and perform validation, you will see a warning message. The warning states that the bundle you are attempting to remove has a dependency on *MyProject-B*.


- Circular Dependencies.** Assume that you have the following bundles in the Bundle Publisher view: *MyProject-A*, *MyProject-B*, and *MyProject-C*. None of the bundles has been published to the server. *MyProject-A* has a dependency on *MyProject-B* and *MyProject-B* has a dependency on *MyProject-C*. *MyProject-C* has a dependency on *MyProject-A*. This represents a circular dependency. Attempts to publish these bundles will fail. If you select the bundles and perform validation, you will see a warning message, describing the following circular dependency: *MyProject-C* has a dependency on *MyProject-B*, which depends on *MyProject-A*, and *MyProject-A* depends on *MyProject-B*.

The following table shows the imports package and the exports package for each bundle.

	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

Refreshing the Bundle Publisher View


➤ To refresh the contents of the Bundle Publisher view

1. In Designer, go to the Bundle Publisher view.
2. Click .

Validating Bundles

In the Bundle Publisher view, you can perform a dependency check on a selection of bundles. The dependency check ensures that you catch potential errors before you attempt to publish a group of bundles to the server. When you perform a dependency check, Application Platform produces a dependency graph for the selected bundles based on the bundles' declared package imports and exports. If the server is started, the dependency check will also include the items, which are currently published to the server. This enables you to check the potential impact of unpublishing bundles from the server.

➤ To validate bundles in the Bundle Publisher view

1. In Designer, go to the Bundle Publisher view.
2. Verify that the **Project** field is cleared.
3. Click .

Bundle Manager View

You can use the Bundle Manager view in order to create bundles from non-OSGi jars, that is, plain jars. When you create a bundle from one or more plain jars, you need an OSGi bundle to host the plain jar. An OSGi bundle that hosts a plain jar, is called a wrapper bundle. A single wrapper bundle is used for publishing one or more plain jars to the server. The Bundle Manager uses different icons to distinguish between plain jars and OSGi bundles.

Creating Wrapper Bundles

Before you can publish plain jar files to the server, you must create one or more wrapper OSGi bundles for the plain jar files.

➤ **To create a wrapper bundle for one or more plain jar files**

1. In Designer, go to the Bundle Manager view.
2. Select the plain jar or jars for which you want to create a wrapper bundle.

If you attempt to create a wrapper bundle for an OSGi bundle file, you will receive an error. You can only create wrapper bundles for plain jar files.

For information about how to add more plain jar files to the Bundle Manager view, see [“Configuring Bundle Manager View” on page 77](#).

3. Click .
4. In the Create an OSGi bundle page of the Create Bundle for Selected Jars dialog specify the provided settings.

The following table describes what you have to specify for each setting.

For this setting	Specify
Bundle Type	<p>One of the following:</p> <ul style="list-style-type: none"> ■ Unwrapped - to unwrap the contents of selected jar(s) in the root directory of generated bundle. ■ Embedded - to embed the selected jar(s) in the generated bundle. <div style="background-color: #f0f0f0; padding: 5px; margin-top: 10px;"> <p>Important: If you have selected more than one jar file, make sure that each jar contains a set of files with unique names to prevent files from one jar to be overwritten by files from the other jar.</p> </div> <p>If you are converting a signed jar file, Software AG recommends that you use the Embedded option to ensure that the integrity of the signed jar is maintained when wrapping the jar as a bundle.</p>
Bundle Symbolic Name	A symbolic name for the bundle.
Bundle Version	A bundle version.
Directory	<p>Select the required directory from the drop-down list box. If the required directory is not available, add the directory from the Bundle Manager settings.</p> <p>For information about adding a directory, see “Configuring Bundle Manager View” on page 77.</p>

5. Optional. Click **Next**. On the Bnd Classpath Settings page select one or more jars to be added to the classpath when building the bundle.

The selected jars are added to the `-classpath` Bnd directive as absolute URIs.

6. Optional. Click **Next**. On the Bnd Template Settings page edit the bundle manifest content before building the bundle.

For information about the syntax and supported options of Bnd templates, see <https://bnd.bndtools.org/>.

7. Optional. Click **Next**. On the OSGi bundle manifest page update the contents of the bundle manifest, if required.

The manifest contents must comply with the general specification for jar manifest files. For information about the jar manifest specification, see the Oracle documentation.

8. Click **Finish**.

Deleting Bundles and Jars

You can use the Bundle Manager view in order to remove OSGi bundle files and plain jar files.

➤ To delete bundles or jars from the Bundle Manager view

1. In Designer, verify that the bundles or jars you want to delete are not published to the server.
2. If there are bundles or jars, which are published to the server, go to the Bundle Publisher view and unpublish them.

You can only unpublish bundles or jars when the relevant server is started in the Servers view of Designer.

For information about unpublishing bundles from the server, see “[Publishing and Unpublishing Bundles](#)” on page 70.

3. Click **X**.

Configuring Application Platform

Application Platform configuration is supported for the following elements:

- Bundle Publisher View
- Bundle Manager View
- Eclipse Capabilities
- Servers View

- Project Configuration
- Customer Applications


Configuring Bundle Publisher View

The configuration of the Bundle Publisher view is divided in separate sections.

Use the **View Contents** section to configure what items to show in the Bundle Publisher view. Note that user bundles are always displayed and you cannot hide them from this section.

Use the **Bundle Dependency Validation** section to limit the amount of content that is returned during bundle validation, or to convey bundle changes to the server.

> To configure the Bundle Publisher view

1. In Designer, go to the Bundle Publisher view, click , and then click **Settings**. Alternatively, go to **Window** menu and click **Preferences**.
2. In the Preferences dialog box, expand **Software AG**.
3. Expand **Application Platform**, and then click **Bundle Publisher**.
4. In the Bundle Publisher page set the provided settings.

The following table describes the check boxes you can set.


Set this check box	To
Plain Jars (not OSGi)	<p>Select to display plain jars in the Bundle Publisher view. When Plain Jars (not OSGi) is cleared, plain jars are not displayed.</p> <div style="background-color: #f0f0f0; padding: 5px;"> <p>Note: Plain jars cannot be published, so you cannot select them in the Bundle Publisher view.</p> </div> <p>Default: check box is cleared</p>
Platform Server Bundles	<p>Select to display the platform server bundles, which are delivered with the server profile, in the Bundle Publisher view. When Platform Server Bundles is cleared, the platform server bundles are not displayed.</p> <p>If you display the platform server bundles, make sure that you do not remove any platform server bundles from the server.</p> <p>Default: check box is cleared</p>
User Bundles	<p>Display all user bundles in the Bundle Publisher view.</p>

Set this check box	To
	Default: check box is always selected
Show Warnings for server bundles	<p>Select to view the OSGi manifest warnings for active server bundles. When Show Warnings for server bundles is cleared, OSGi manifest warnings are not displayed.</p> <p>Note: Selecting this check box may produce excessive warning messages in unrelated bundles when publishing or validating bundles.</p> <p>Default: check box is cleared</p>
Show warnings for missing optional imports	<p>Select to view warning messages when imported packages for a bundle are not included. When Show warnings for missing optional imports is cleared, warning messages for missing imported packages for bundles are not displayed.</p> <p>These warning messages are displayed when the Bundle Publisher is validating bundles or applying updates to the server. For example, if one of the bundles in the set has a <code>manifest.mf</code> file containing an <code>Import-Package</code> header with the <code>resolution := optional</code> qualifier, and no active bundle exports exist for this package, a warning message will be returned if this check box is selected.</p> <p>Note: In some cases a missing package does not result in a warning message, for example when you are performing imports to a test framework.</p> <p>Default: check box is cleared</p>

Configuring Bundle Manager View

Use the Bundle Manager view to define one or more directories that contain additional bundles to be published to the server. You can share bundles that reside in one of these directories across your projects.

» To configure a directory for additional bundles in the Bundle Manager view

1. In Designer, go to the Bundle Manager view, click , and then click **Settings**. Alternatively, go to **Window** menu and click **Preferences**.
2. In the Preferences dialog box expand **Software AG**, then expand **Application Platform**, and then click **Bundle Manager**.
3. Click **Add Directory**.

4. Select the directory you require and click **OK**.
5. Click **Apply**, and then click **OK**.

Defining Application Platform Capabilities

Use the Eclipse capabilities to associate a collection of views or activities to a specific purpose. After you define a capability, you can use it to quickly hide those related items. Application Platform also provides custom capabilities.

> To configure Application Platform capabilities

1. In Designer, go to **Window** menu and click **Preferences**.
2. In the Preferences dialog box, expand the **General** menu and click **Capabilities**.
3. On the Capabilities page, specify values in the provided fields.

The following table describes what you have to specify for each field.

In this field	Specify
Prompt when enabling capabilities	When this check box is selected, you will be prompted to confirm enabling the capabilities. By default, this check box is cleared.
Capabilities	The list of the groups of capabilities you can configure. Here you can enable or disable the Software AG App Platform capability. By default, the Software AG App Platform capability is enabled.
Description	A description for the selected capability.
Requires	If the selected capability, requires enabling other capabilities, they are listed here.

4. Click **Advanced...** to select the next configuration view.
5. Expand **Software AG App Platform** and configure the provided Application Platform capabilities.

The following table describes the capabilities that you can configure.

Capability	Description
App Platform Core	The core development features of Application Platform, used for developing Java projects.

Capability	Description
	By default, this capability is enabled.
App Platform Web	The web development features of Application Platform, used for developing web projects. By default, this capability is enabled.
Integration Server Extensions	All Integration Server-related features of Application Platform. By default, this capability is enabled.

6. In the Advanced Capabilities Settings dialog box, click **OK**.
7. On the Capabilities page click **Apply**, and then click **OK**.

Configuring Servers View

You can update the configuration of the Servers view.

➤ To configure the Servers view

1. In Designer, go to **Window** menu, click **Preferences**.
2. In the Preferences dialog box, click **Server**.
3. Select the **Show Servers view when server state changes** check box if you want the Servers view to be activated whenever there is a server activity, including startup, shutdown, or project changes.
4. In the Navigation panel, expand the **Server** menu and click **Launching**. Use this page to configure the server launching settings.

For information about the launching configurations, see the *Web Tools Platform User Guide in Software AG Designer Online Help*.

Configuring Application Platform Projects

Projects that you create in Designer contain project-specific properties. Application Platform projects contain the Application Platform Core project facet and an Application Platform project property configuration.

You configure the properties of your Application Platform projects in Designer.

➤ To configure Application Platform projects

1. Right-click the required project and click **Properties**.
2. In the Properties dialog box, expand **Application Platform**.
3. Click **Project Version**.
4. In the **Version** field enter the project version.

The version string must be valid for the OSGi standard and it must include three numeric values, separated by periods. Bundles you create for this project will adopt the same value for the *Bundle Version* manifest header.

Note:

If you have created a custom Bnd template using the **Create Bnd template** tool from the project context menu, you must update the Bnd template with the updated project version. This will ensure that the *Bundle Version* header is up to date.

Important:

If you are deploying your project using Deployer, you must create a project manifest in order to use the **Project Version** property. You must include the manifest with other project files that you commit to source control, so that the manifest is available when using the Asset Build Environment. This will ensure that the bundle produced by the Asset Build Environment contains the expected *Bundle Version* property.

5. Optional. If you are configuring a web project, go to the Navigation panel and click **Project Bundle**. Fill in the **Web Context Path** field.

You can update the web context path of projects, created with the Application Platform Web project facet, after creating the project.

Developing Custom Applications

Applications you develop may include properties files, which contain key-value pairs that allow you to configure the values on each server where your application is deployed. Application Platform expects these key-value pairs to be implemented as properties files. This section explains how the properties files are created and installed on the server.

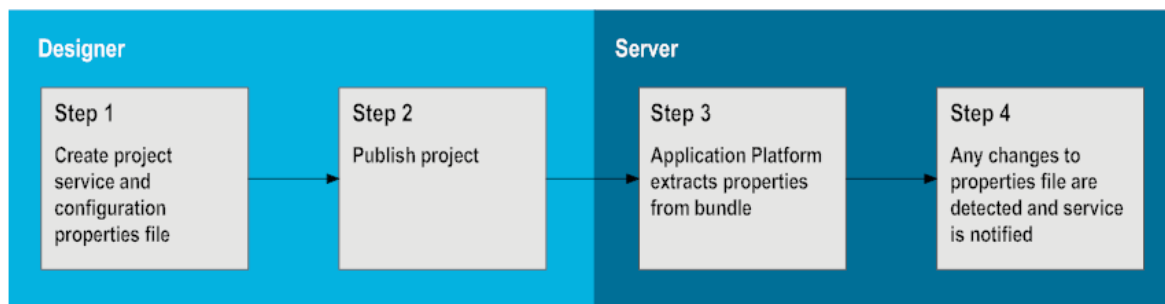
The following rules apply to the properties files:

- You must include all properties files in the `src/main/config` folder of the associated project.
- You must add a unique name to each properties file.
- You must name the files following a reverse domain name convention. For example, company XYZ might have a `com.xyz.demo.dataSource.properties` file.
- You must begin the names of the files for internal use with “`com.softwareag.`”. These files are not deployed to the server.

- You must not share properties files across projects because all properties files are removed when a project is unpublished.

The following diagram illustrates the steps for managing configuration data while in Designer. The steps are as follows:

1. Create project service and configuration properties file. This step is executed in Designer.
2. Publish the project. This step is executed in Designer.
3. Application Platform extracts properties from the bundle. This step is executed on the server.
4. Any changes to the properties file are detected and the service is notified. This step is executed on the server.



➤ To develop custom applications

1. Open Designer and create a properties file in the `src/main/config` directory. Implement a Java class to use these configuration properties.

Note:

When you unpublish your project from the server using the Servers view in Designer, the affected properties file will be removed from the server.

2. Publish the project to the server.

When you publish the project in Designer, a bundle is 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 project manifest. The configuration properties files contained in the project override any properties files that reside on the server. The server responds to any subsequent edits in the properties file by notifying the managed service. For more information, see *Getting Started with the Application Platform API*.

When a project bundle is created, Application Platform checks the bundle for the special `AP-Bundle-ConfigFiles` header. If Application Platform finds this header, it extracts all listed properties files and installs them to the server profile's directory for dynamic configuration, located here: `Software AG_directory \profiles\server_instance\configuration\com.softwareag.platform.config.propsloader`.

Important:

Do not remove the `AP-Bundle-ConfigFiles` property header from the bundle. This header is only produced when projects are published in Designer. The Asset Build Environment tool does not create this header when it produces a project bundle.

About Using Services in Application Platform

This section describes how you can browse and expose services from your Application Platform projects.

You can browse services from your Application Platform projects by using the Service Browser view, available in the Application Platform perspective.

You can expose services from and to Application Platform projects. Currently, the following scenarios are available:

- Calling Integration Server services from Application Platform projects.
- Calling Application Platform services from Integration Server services.

You can generate Java service wrappers in Application Platform projects, using native Integration Server data structures. Currently the following scenarios are available:

- Using Integration Server Document Types in the input and output signatures of a service.
- Using Integration Server Specifications in the input and output signatures of a service.

Service Browser View

You can use the Service Browser view in order to browse the services used in your project bundles. When you publish a project to the server, all the services that are used in that bundle are listed in the Service Browser view. The Service Browser view displays all services that are published to the currently selected server in the Servers view. To switch between servers, you must change the selection in the Servers view. You can filter the services by service name or project bundle. You can also customize the Service Browser view by selecting the filters to be used and the content it displays.

Searching in the Service Browser View

In the Service Browser view you can search for services by using the search bar.

➤ To search for services in the Service Browser view

1. In Designer, go to the Service Browser view.
2. Type the required text in the search bar.



You can search by service name, service details, bundle name, or bundle details.

3. Press Enter.

Grouping Services by Bundle Name

By default, the services displayed in the Service Browser view are grouped by service name. You can also group the services by the name of the bundle that has registered the service.


➤ To group services by bundle name

1. In Designer, go to the Service Browser view.
2. Click .
3. Optional. Click  again to group the services by name.

Refreshing Services and Bundle Information

In the Service Browser view you can refresh the services and the bundle information that is published to your runtime environment.


➤ To refresh services and bundle information

1. In Designer, go to the Service Browser view.
2. Click .

Filtering Services Displayed in the Service Browser View

You can filter the services displayed in the Service Browser view by using the available custom filters.

➤ To filter the services displayed in the Service Browser view

1. In Designer, go to the Service Browser view.
2. Click .
3. Click **Customize View**.
4. In the **Filters** tab set the provided check boxes to define the customizing filters to be used.

The following table describes what services you can filter by setting the available check boxes.

Set this check box	To filter
Apache namespace services	Services that start with org.apache.

Set this check box	To filter
App Platform Secure Services	Services that are registered by POJO classes annotated with both @Secure and @Service.
App Platform Services	Services that are registered by POJO classes annotated with @Service.
App Platform Services published to Integration Server	Services that are exposed to Integration Server with the @ExposeToIS annotation.
Eclipse Services	OSGi services that start with org.eclipse.
Java namespace services	OSGi services that start with java or javax.
OSGi Services	OSGi services that start with org.osgi.
Platform Services (registered by Software AG bundles)	OSGi services that are registered by internal Software AG product bundles. The parent bundles of these services have symbolic names that start with com.softwareag.
Software AG Platform Services	OSGi services that start with com.softwareag.

Customizing Content Displayed by the Service Browser View

You can customize the content that the Service Browser view displays.

➤ To customize content, displayed by the Service Browser view

1. In Designer, go to the Service Browser view.
2. Click ▾.
3. Click **Customize View**.
4. Click the **Content** tab.
5. Set the provided check boxes to define the content to be displayed.

The following table describes what you can display by setting the available check boxes.

Set this check box	To display
Runtime services	The full list of services that are published to the runtime. If you disable this check box when services are grouped by service name, you will hide all services.
Bundle details	Details about each bundle when the services are grouped by bundles. The following details are displayed:

Set this check box	To display
	<ul style="list-style-type: none"> ■ Bundle headers ■ Export package details ■ Import package details <p>If you disable this check box when services are grouped by bundles, you will hide all bundle details.</p>
Service details	<p>Details about each service when services are grouped by service name. The following details are displayed:</p> <ul style="list-style-type: none"> ■ Service properties ■ Name and version of the bundle that has registered the service <p>If you disable this check box when services are grouped by service name, you will hide all service details.</p>
Runtime bundles	<p>The bundles that have registered the services. If you disable this check box when services are grouped by bundles, you will hide all services.</p>

Calling Application Platform Services from Integration Server Services

Integration Server allows you to have Integration Server (IS) services that call Java source files in Application Platform by attaching annotations to methods in Application Platform. When you publish projects that contain these annotated methods to Integration Server, IS service bindings are created, which can be invoked in Integration Server flow services, or executed by Integration Server Java services.

When you expose Java methods to Integration Server, you must use annotations in order to mark the specific method(s) to expose. This section provides an overview of the required functional steps.

For more information, see *Getting Started with the webMethods Application Platform API*.

➤ To call Application Platform services from IS services

1. Annotate a method by marking the class and method with the necessary annotations.

The following table describes the available methods.

Method	Description
@Service	Use the @Service class annotation to identify the class as a service, so that it can be included in the server's OSGi service registry.
@ExposeToIS	Use the @ExposeToIS class annotation to provide additional details for Integration Server.
@ExposedMethod	Use the @ExposedMethod method annotation to identify the method to be used when creating an IS service.

Example:

```
@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) {
```

2. Publish your project.

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

3. Verify that the Integration Server package you provided in step 1 with the @ExposeToIS annotation exists, and that it contains the proper service signatures.

Coding Considerations

There are some architectural differences between coding with Java and coding with Integration Server. You must keep the following points in mind when you call Application Platform services from IS services:

- IS services are stateless operations on a pipeline. Holding references to Java objects in the Integration Server pipeline is not supported. Make sure that the exposed operations do not depend on a Java objects' holding state.
- Application Platform and Integration Server use different class loaders, so object references are not transferred between them. Java objects that you use in a method signature must be Java Beans. The IS services that are generated will include signatures that use objects of type Document in order to represent the Java Bean objects.

Calling Integration Server Services from Application Platform Projects

You can expose your Integration Server Java services and Integration Server flow services to your Application Platform projects. Java source file binding classes are code-generated to facilitate calling the IS services.

In Application Platform, you can create Java source files that are essentially client stubs used to invoke Integration Server Java services and Integration Server flow services from an Application Platform project.

For more information, see *Getting Started with the webMethods Application Platform API*.

➤ To call Integration Server services from your Application Platform project:

1. Right-click your project and select **App Platform**, and then select **IS Tools**.
2. Click **IS Service Wizard**.


Note:

You can also launch the **IS Service Wizard** by pressing Ctrl+Shift+Z.

3. Select a destination project from the **Project** drop-down list box.

You can select only projects with an enabled IS Service Extensions project facet.

4. Select the required IS services. If you see no IS services, or less IS services than you expect, try one of the following:

- Right-click the required node and click .
- Right-click the required node and click **Refresh tree contents**.

You can select one or more IS services. You must use only user-developed services with valid IO specifications.

Note:

None of the Integration Server product services, contained in packages that begin with Wm*, are visible in the wizard. Therefore, you will see no services for selection before you have created at least one custom service.

5. Click **Finish**.

Java bindings are created for the selected IS services. You can find the source files in the source directory, as defined for the IS Service Extensions project facet. The default location is genSource. The package name is determined from the IS service and its parent folder name(s). Each IS service you select has its own dedicated Java package. This ensures that there is no overlap with the generated input and output classes.

Important:

If you deploy a project that calls IS services to the production server, you must set the generated source directory to `src/main/java`.

Coding Considerations

There are some architectural differences between coding with Java and coding with IS services. You must keep the following points in mind when you call IS services from your Application Platform projects:

- While Java is Object-Oriented, IS services are stateless operations on a pipeline. When you generate a Java class 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 a map.
- While Java methods are defined in classes which are in packages, IS services are defined in folders and in Integration Server packages. Java packages and Integration Server packages are different concepts. A class's Java package path uniquely represents the class in the Java class namespace. An Integration Server package is a unit of packaging but it is not part of the service namespace. In the Integration Server namespace, the folder and service name uniquely identify a service. The folder name in the Integration Server namespace is usually a dot-separated list of words, for example: *this.is.my.folder.name*. Also, it is not uncommon that Integration Server folders include capital letters, while Java packages are almost always lowercase. Application Platform combines the Integration Server folder and service names to create a Java package name.
- Java and Integration Server use different data types. Java's data type system is very rich, it includes primitive types and every class ever created. Integration Server has a much smaller data type system. Integration Server supports String and Java primitive wrapper types. However, complex structures in IS services are typically modeled using the 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. An IS document with nested Document elements is similar to a map that represents properties in a Java Bean. Application Platform takes advantage of this similarity. The input and output classes it generates are simple Java Beans with a property that represents each input or output value. If the IS service input signature includes a Document type, then a Java (Bean) class is generated to represent the Document structure.

The following table shows the data type mapping between Java data types and Integration Server data types.

Java Data Type	Integration Server Data Type
<code>java.lang.String</code>	String
Primitives; int, float, and other	Object->Primitive Wrapper; Integer, Float, and other
Primitive Wrappers; <code>java.lang.Integer</code> ,	Object->Primitive Wrapper; Integer, Float, and other

Java Data Type	Integration Server Data Type
java.lang.Float, and other	
java.util.Date	Object->Primitive Wrapper; Date
Java Bean class	Document A map of property name => property value, where String and primitive properties are represented as described here, and other types are represented as nested Documents.

- Application Platform and Integration Server 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 both Integration Server and Java. More complex object structures are represented by Java Beans in Java and by IS documents in Integration Server. The list of elements with similar representation includes a byte array, so you can pass serialized objects if you handle serialization in Application Platform and ensure that appropriate classes are available on both sides.
- Java and Integration Server recognize different words as having special meaning. For example, an IS service can have an 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 Application Platform code generator adds `PLS_` or `pls_` in front of generated class and property names. However, there still can be cases when the generated code does not compile properly. In such cases, try to use flow mapping in Application Platform in order to change parameter or service names.
- Java and Integration Server recognize different sets of characters as having special meaning. An IS service and parameter names can use `@`, `*`, or other characters that are not allowed in Java class names and variable names. You should avoid such conflicts by changing service names and property names in Integration Server.
- The Application Platform code generator relies on the IS service to have an explicit service signature that defines all input and output elements. Such a signature is not required in Integration Server. If you want to call an IS service without a signature from Java and it is not possible to add a signature to this IS service, you must create a flow wrapper that has an appropriate signature and invokes the service.

Generating POJO Wrappers for IS Document Types and Specifications

When using Integration Server services with Application Platform projects, Application Platform generates the input and output signatures of the services inline. Instead of using these inline references with individual services, you can generate Plain Old Java Object (POJO) classes that wrap Integration Server native data structures, such as IS document types and specifications, and publish the generated wrapper packages to Integration Server. With POJOs, multiple Application Platform projects can reference those packages as dependencies.

➤ **To generate POJO wrappers for IS document types and specifications:**

1. In the Package Explorer view, right-click your project and select **App Platform > IS Tools > IS Service Wizard**.

Tip:

You can start the **IS Service Wizard** by pressing Ctrl+Shift+Z.

2. Select a destination project from the **Project** drop-down list.

You can select only projects for which the IS Service Extensions project facet is enabled.

3. Select the required IS document type or specification reference.
4. Click **Finish**.

Application Platform creates a package that contains the generated POJO wrapper for the IS document type or specification. You can find the package in the source directory, defined for the IS Service Extensions project facet. The default location is `/genSource`. The name of the package is: `pub.parent_folder_name.reference_name`.

Coding Considerations

You must keep the following points in mind when generating POJO wrappers for IS document types and specifications:

- You cannot generate service wrappers for product services from Wm packages, but you can generate POJO wrappers for IS document types and specifications, residing in Wm packages and then reference these data structures from custom services.
- You can use nested document types, but you cannot generate a service wrapper for a document type that includes a recursive reference, for example, when a field in an IS document type refers to itself.
- You can use document types and specifications that contain object references only to String, Date and primitive data types. You cannot generate a service wrapper for a document or specification that contains a reference to an IS IData object.
- The valid identifiers for field and record names in Integration Server differ from valid Java identifiers. To address this when generating identifiers for Java services, Application Platform substitutes characters, that are illegal in Java with underscores.

Application Platform Tutorial

In addition to *webMethods Application Platform User's Guide* and *Getting Started with the webMethods Application Platform API*, there is also a *webMethods Application Platform Tutorial*. The *webMethods Application Platform Tutorial* provides step-by-step examples for performing common tasks in Application Platform. It also provides sample projects, which allow you to practice developing in

Application Platform. You can download the *webMethods Application Platform Tutorial*, together with the code samples, from the Software AG TECHcommunity.

3 Working with Application Platform Projects

- About Deploying Projects 94
- About Configuring Published Projects 97

This topic describes the tasks, which you can execute after your Application Platform project is implemented and you want to deploy the project to the server.

About Deploying Projects

You can deploy your Application Platform projects with webMethods Deployer. Deployer ensures that reproducible builds of your developed projects are produced outside of Designer. This section provides a high-level description of the steps you need to execute in order to deploy a project. For more information, see *webMethods Deployer User's Guide*.

Before using Deployer, you must prepare your project bundles with the Asset Build Environment command line tool. With Asset Build Environment you can create bundles from the source of your Application Platform projects. Deployer then uses the generated bundles for deploying your projects. In Deployer the bundle files are referred to as *assets*. For more information about Asset Build Environment and Deployer, see *webMethods Deployer User's Guide*.

Before you can create bundles for an Application Platform project with the Asset Build Environment, you must do the following configurations:

- Configure a properties file for the Asset Build Environment.
- Configure a properties file for your Application Platform project.

Configuring Asset Build Environment

Asset Build Environment has its own properties file, which includes properties that control which products are included, the location of the project source files, and so on.

➤ To configure the Asset Build Environment properties file for your Application Platform project

1. Go to the *Software AG_directory* \common\AssetBuildEnvironment\master_build directory.
2. Open the *build.properties* file.
3. Add the necessary configurations for your Application Platform project.

Configuring Application Platform Projects for Asset Build Environment

Designer creates a properties file for each Application Platform project. Designer adds this file to the project's root folder.

➤ To configure the Application Platform project properties file for Asset Build Environment

1. Go to the root directory of your Application Platform project.

2. Open the `assetBuild.properties` file in a text editor.

For a detailed description of the contents of the `assetBuild.properties` file, see “ [Application Platform Project Configuration for Asset Build Environment](#) ” on page 95.

3. Commit the `assetBuild.properties` file, together with the remaining project source files, to the Asset Build Environment.

Application Platform Project Configuration for Asset Build Environment

The following table describes the property names, contained in the `assetBuild.properties`, generated by Designer. The table contains the property names, their value type and description, and it states if the property is required.

Property Name	Value Type	Description	Required
<code>component.name</code>	String	This name is used for the bundle file name and the following manifest headers: Bundle-Name:	Yes
<code>component.type</code>	String	For Application Platform this value must always be <i>bundle</i> .	Yes
<code>component.home</code>	String	The path to the project source.	No, Asset Build Environment will assume that a project with <code>component.name</code> is located in <code>master_build/build.properties</code> <code>build.source.projects</code>
<code>component.dependencies</code>	Comma delimited String	A list of component names to be included on the classpath when building this project.	No
<code>component.webcontext</code>		This property is reserved for future use.	No, the web context path defaults to the project name but it can be overridden by the Web-ContextPath: OSGi

Property Name	Value Type	Description	Required
			manifest header.
<code>component.version</code>		This property is reserved for future use.	No, specify the bundle version in the project's <code>bnd.bnd</code> file with the <code>Bundle-Version:</code> header.
<code>component.src.dirs</code>		This property is reserved for future use.	No, all source files must be in the <code>src/main/java</code> directory.
<code>component.dependencies.external</code>	String	This property is a reserved value that is automatically generated in the project's <code>assetBuild.properties</code> file. Its value is a file pattern relative to the configured <code>BUILD_EXTERNAL_DIR</code> classpath variable in Designer and relative to the <code>build.external.dir</code> Ant property defined in the master build properties file of Asset Build Environment. For example, this value can be <code>com/springsource/3.2.3/*.jar</code> .	No

Creating Assets with Asset Build Environment

To deploy bundles with Asset Build Environment and create assets, you must embed the bundles in the project. For this purpose, you must include the bundles in the `lib` directory of the project. In this way the bundles are part of that project bundle's classpath. When you do this, make sure that every project that has a dependency on one or more bundles, includes the bundles in its own directory.

Deploying Assets in Deployer

After you create assets using the Asset Build Environment, you can use Deployer to install the assets to the target servers. For information about deploying assets, see *webMethods Deployer User's Guide*.

About Configuring Published Projects

Your Application Platform projects can contain configuration data, which is included when you publish a project to the server. After a project is on the server, you can modify this configuration data dynamically during runtime.

Using the Project Dynamic Configuration

The Application Platform runtime allows you to configure project properties dynamically through the ConfigurationAdmin service of Software AG Common Platform.

Before you start using the project dynamic configuration, keep in mind that in traditional Java projects configuration files, such as properties files, are loaded using the `class/classloader` of the currently executing method or thread. The configuration files are present in the classpath of the running program and they are accessible during runtime either with the project bundle that is published, or globally as part of the runtime container. However, if you are using this approach, you have to republish the project bundle, or restart the runtime container before you can update dynamically the properties file with an external service, for example the ConfigurationAdmin service of Software AG Common Platform.

To support dynamic updates in Application Platform project configuration, ensure the following during project development:

- Keep the project's properties file in the `src/main/config` directory of the project.
- Add a unique name to the project's properties file. This name is used as a *persistent identifier (PID)* that identifies the properties file.

For more information about the properties file, see [“Developing Custom Applications” on page 80](#).

- Ensure that classes that need to dynamically update the properties file implement the OSGi `org.osgi.service.cm.ManagedService` interface and the associated updated `Map` `properties` callback method.
- Ensure that classes that need to dynamically update the properties file are published as managed services, so that they can receive notifications about configuration file changes. For this purpose, you must use the `@Service` annotation, which allows you to publish a class as an OSGi service. In the annotation, specify the following type as one of the exported interfaces:
`org.osgi.service.cm.ManagedService`.

Note that the project's properties file is packaged with the published bundle in the container but it is extracted and stored in the following location of the common configuration store of the installed

runtime: *Software AG_directory* \profiles\IS_default\configuration\
com.softwareag.platform.config.propsloader.

For more information, see *Getting Started with the Application Platform API*.

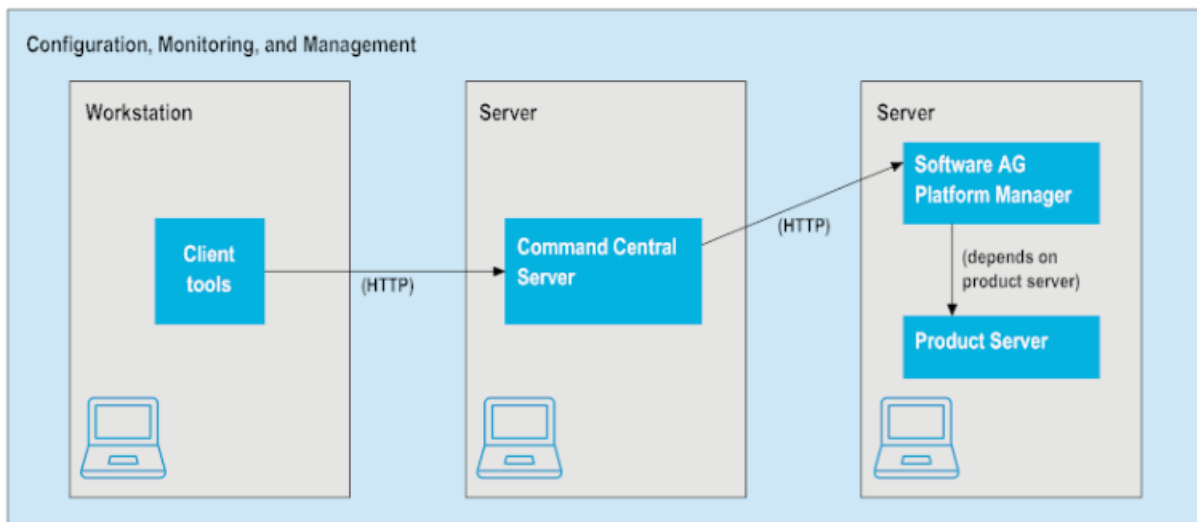
4 Administering Application Platform Using Command Central

- Managing Application Platform Projects Using Command Central 100
- Monitoring KPI Data for WAR Projects Using Command Central 100

Managing Application Platform Projects Using Command Central

Software AG Platform Manager and Software AG Command Central provide a common infrastructure for managing your product configuration and monitoring your product statuses. Application Platform is also managed in this common infrastructure by a dedicated server, called Platform Manager. You can monitor Application Platform by using Command Central. Command Central allows you to administer Software AG products across networked servers through a command line interface or through a web-based user interface.

The following diagram illustrates the configuration, monitoring, and management of Application Platform projects using Software AG Command Central and Software AG Platform Manager. The diagram consists of three components: a workstation component and two server components. The workstation component contains client tools and it communicates to the first server component through HTTP. The first server component contains a Command Central server. It also communicates through HTTP to the second server component. Finally, the second server component contains Platform Manager, which depends on a product server.



For more information about Platform Manager and Command Central, see *Software AG Command Central Help*.

For information about migrating Application Platform projects using Command Central, see the *Upgrading Software AG Products On Premises* guide.

Monitoring KPI Data for WAR Projects Using Command Central

You can use Command Central to monitor KPI data for your WAR projects that are developed in Application Platform and deployed to Integration Server. You can view the KPI data through the Command Central CLI.

For example, to get the run-time status and run-time state for a component with ID "OSGI-IS_default-AppPlatform" running on the local installation, and all its dependent components:

```
sagcc get monitoring state nodeAlias=local  
runtimeComponentId=OSGI-IS_default-AppPlatform
```


5 Diagnostics and Troubleshooting

■ Useful Logs for Application Platform	104
■ Increasing Tomcat Debug Logging	104
■ Using Log4j in WAR projects	106
■ JSP Validation in WmAppPlat	107
■ Diagnosing Bundles with the OSGi Console	107
■ Considerations When Publishing Projects to Servers	107
■ Common Project Issues	108

This topic describes how you can diagnose and troubleshoot Application Platform issues in Designer and on the server.

Useful Logs for Application Platform

You can use the following logs to diagnose Application Platform issues:

- **Designer Log Files.** The Designer log file is located here: `Workspace_Directory\.metadata\.log`
- **Designer Trace Logging.** Designer includes a tracing convention for capturing additional content when something goes wrong. Application Platform supports this convention too. In order to use trace logging with Application Platform, you must perform additional configuration steps. For information about trace logging configuration for Application Platform, see the *Workbench User's Guide* in *Software AG Designer Online Help*.
- **Server Log Files.** The server produces several log files. The log files for Integration Server's default instance are located here: `Software AG_directory\profiles\IS_default\logs\`. The available log files are `wrapper.log` and `sag-osgi.log`.
- **Configure Server Debug Output.** You can configure additional debug output for the server. For Integration Server's default instance, use the following location: `Software AG_directory\profiles\IS_default\configuration\logging\log4j2.properties`. In the `log4j2.properties` file add the following lines:

```
logger.n.name=com.softwareag.applatform.pls.is.web
logger.n.additivity=true
logger.n.level=debug
```

where *n* is the next available key number.

Restart the server after updating the `log4j2.properties` file. This formatter captures Application Platform debug messages.

For more information, see [“Software AG Servers” on page 14](#).

Increasing Tomcat Debug Logging

When you are debugging issues related to the WmAppPlat package, it might be useful to increase the Tomcat debug logging levels. Software AG recommends that you add the following trace log level groups:

Tomcat Container Startup

```
<!-- server startup -->

<logger name="org.apache.catalina.core.StandardServer" additivity="true">
  <level value="debug" />
</logger>

<logger name="org.apache.catalina.core.StandardContainer" additivity="true">
  <level value="debug" />
</logger>
```



```

<logger name="org.eclipse.gemini.web.tomcat.internal.OsgiAwareEmbeddedTomcat"
  additivity="true">
  <level value="debug" />
</logger>

<logger name="org.eclipse.gemini.web.tomcat.internal.StandardWebApplication"
  additivity="true">
  <level value="debug" />
</logger>

```

War Deployment

```

<!-- war deployment -->

<logger name="org.apache.catalina.core.StandardContext" additivity="true">
  <level value="debug" />
</logger>

<logger name="org.eclipse.gemini.web.tomcat.internal.ExtendedStandardContext"
  additivity="true">
  <level value="debug" />
</logger>

<logger name="org.apache.catalina.core.ContainerBase" additivity="true">
  <level value="debug" />
</logger>

<logger name="org.apache.catalina.startup.ContextConfig" additivity="true">
  <level value="debug" />
</logger>

<logger name="org.apache.catalina.core.StandardContext" additivity="true">
  <level value="debug" />
</logger>

```

Security

```

<!-- security -->
  <logger name="com.softwareag.platform.catalina.auth" additivity="true">
    <level value="debug" />
  </logger>
  <logger name="org.apache.catalina.realm" additivity="true">
    <level value="debug" />
  </logger>

  <logger name="com.softwareag.security.sin.jaas.util.JaasConfigUtil"
additivity="true">
    <level value="debug" />
  </logger>
  <logger name="com.softwareag.security.authz.shiro.AuthorizationServiceImpl"
  additivity="true">
    <level value="debug" />
  </logger>
  <logger name="com.softwareag.security.jaas.login.modules.SAMLAssertionLoginModule"
  additivity="true">

```

```
    <level value="debug" />
  </logger>
  <logger name="com.softwareag.platform.wab.filter.saml.SAMLAAssertionFilter"
additivity="true">
    <level value="debug" />
  </logger>
  <logger name="org.apache.shiro.subject.support.DelegatingSubject" additivity="true">
    <level value="debug" />
  </logger>
  <logger name="org.apache.catalina.authenticator.BasicAuthenticator"
additivity="true">
    <level value="debug" />
  </logger>

  <logger name="com.softwareag.platform.catalina.auth.SINRealm" additivity="true">
    <level value="debug" />
  </logger>
```

JSP Invocation

```
<!-- jsp invocation and AP listeners -->

  <logger name="org.apache.jasper.servlet.JspServlet" additivity="true">
    <level value="debug" />
  </logger>
  <logger name="com.softwareag.applatform.pls.is.web" additivity="true">
    <level value="debug" />
  </logger>
```

Tag Library

```
<!-- tag library processing -->

  <!-- Produces debug messages to stdout in wrapper.log for the WmAppPlat IS package
-->
  <logger name="com.softwareag.applatform.pls.is.web" additivity="true">
    <level value="debug" />
  </logger>
  <logger name="org.apache.jasper.servlet.TldScanner" additivity="true">
    <level value="debug" />
  </logger>
  <logger name="org.apache.jasper.servlet.TldScanner" additivity="true">
    <level value="debug" />
  </logger>
  <logger name="org.apache.tomcat.util.scan.StandardJarScanner" additivity="true">
    <level value="debug" />
  </logger>
```

For more information about debug logging in Tomcat, see the Apache Tomcat documentation.

Using Log4j in WAR projects

All WAR projects developed in Application Platform have access to the Apache Commons Logging 1.2 jar, since it is included in the classpath of the Application Platform web-is.jar bundle. You can provide a log4j.properties file with your WAR projects to control logging in the Integration Server log file, located in *SoftwareAG_directory/profiles/IS_instance/logs/sag-osgi.log*.

You can control logging by providing category loggers, for example:

```
<logger name="com.softwareag.war.greet.web"additivity="true"><level
value="debug"/></logger>
```

You can add category loggers by:

- Adding them to the profile logging configuration, located in *SoftwareAG_directory/profiles/IS_instance/configuration/logging/log4j.properties*.
- Providing them directly with the deployed WAR project by including a *log4j.properties* file in *WEB-INF/classes*.

For more information about the *log4j.properties* file, see the Apache Tomcat documentation.

JSP Validation in WmAppPlat

When performing JSP validation, the WmAppPlat ignores custom *webMethods* tags with missing required attributes and logs warning messages for each missing attribute. You can enable exception messages to be logged for missing attributes by adding the following line to *SoftwareAG_directory/profiles/IS_instance/configuration/config.ini*:

```
com.softwareag.applatform.pls.web.is.strictValidation = "true"
```

This configuration enables `javax.servlet.jsp.JspException` exceptions when a required attribute is missing.

Diagnosing Bundles with the OSGi Console

Designer provides a host OSGi console in the Console view. You can use this console to examine the status of bundles installed to the JVM. You can also configure the server to allow access to an OSGi console.

For detailed steps about enabling the OSGi console in the Terminal view, see [“Enabling the OSGi Console” on page 23](#).

Considerations When Publishing Projects to Servers

This section describes common problems that may occur when you publish your Application Platform projects to the server. The following points describe common server issues and recommendation for troubleshooting each issue:

- **Server is installed as a service.** Software AG recommends that when you set up a development environment with Application Platform, you install the server as an *application*, as opposed to a *service*. This is required because the service wrapper scripts do not start the server with the expected configuration, which can lead to a mismatched configuration between Designer and the server.

For example, the service is started without a configured and opened JPDA debugging port. If this happens, try to resolve the problem by stopping the server and restarting it in Designer.

- **Server immediately fails to start.** If the server fails with an error in the Servers view in Designer immediately after startup, confirm that the server startup script is running synchronously. Also, check the server's runtime environment and verify that `BLOCKING_SCRIPT` is not set to `yes`.
- **Server fails to start after the timeout.** If the server fails to start after the timeout, verify that the primary HTTP port number, configured in the Servers view, matches the port number, configured for the server instance. For Integration Server, also verify that valid user credentials exist for the matching Integration Server port configuration found in Designer, under Window > Preferences > Software AG > Integration Servers.

Common Project Issues

This section describes some of the more common issues that can occur when you create and publish project bundles.

Unable to Add a Project to the Server

If you cannot add a project to the server, check if any of the Application Platform project facets are missing. The Application Platform core facet is required for adding a project to the server. Make sure that you use the Application Platform project wizards when you are building a new project.

Unable to Create a Bundle

If you are unable to create a bundle, verify that your project does not contain Java source files in the default package. Currently, such projects are not supported. Make sure that your source files have a qualified package name.

References to Local Resources

Keep in mind that if your projects use traditional Java programming techniques that rely on access to metadata files, you cannot reference your projects remotely across bundle boundaries. Such techniques are Java service provider and thread context classloader.

Unable to Publish Any Project Bundle

If you are unable to publish any project bundle, verify the following conditions:

- The Runtime Environment configuration is stored in a file under the workspace. If you use multiple workspaces with multiple installations of Application Platform, this can lead to confusion. Verify that the Runtime Environment directory path is correct for the current workspace. If Designer is installed in `C:\SoftwareAG`, for example, and its workspace is in `C:\dev\workspace_98`, make sure that the Runtime Environment configuration indicates `C:\SoftwareAG` for the installation home.

- The port number, defined for the server configuration in Designer matches the port number, defined in the server profile configuration.
- The server and the JMX service in the server are started successfully. Verify that the JMX port is in LISTEN mode for the server. If it is not in LISTEN mode, restart the server and check if this issue is resolved.
- The Application Platform components are installed for each server. If the components are not installed, install the missing components and restart the server.

Manually Uninstall a Bundle from the Server

If you delete a project published to the server from Designer, the project bundle will be orphaned. You must remove orphaned bundles manually from the server.

➤ To manually uninstall a bundle from the server

1. While the server is started, delete the orphaned file from the repository directory, located here: *Software AG_directory* \profiles\IS_default\workspace\app-platform\deployer\bundles.

This will ensure that the bundle is not re-deployed the next time you restart the server.

2. Open an OSGi console to the server and uninstall the bundle using its bundle ID.

For information about configuring an OSGi console, see [“Enabling the OSGi Console” on page 23](#).

Class Loader Issues in Published Projects

Careful inspection of the stack trace can provide you with helpful hints about Application Platform issues. The following points describe the most common cases:

- **ClassNotFoundException.** A `java.lang.ClassNotFoundException` usually indicates that a class within the bundle fails to instantiate a class, for example, `Class.forName`, for one of the following reasons:
 - The class is not in the bundle that is raising the exception.
 - The class is not exported by any other bundle on the server.
 - The class is exported by a bundle on the server but the bundle that is raising the exception does not import the class.

Usually, this issue will not be caught when your project is published, as long as there is a source code reference to the class.

For more information, see [“Assembling Project Bundles” on page 63](#).

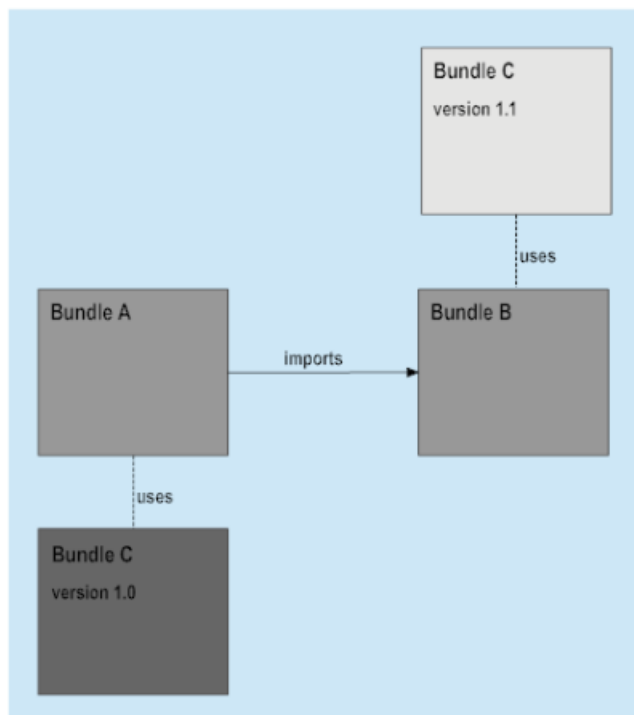
Note:

Jars that are added to a project's classpath through 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.

- **NoClassDefFoundError.** A `java.lang.NoClassDefFoundError` usually indicates that a class that is loaded in the bundle's classloader fails to reference another class while executing.
- **LinkageError and ClassCastException.** These errors usually indicate 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. You must 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 a `java.lang.LinkageError` or a `java.lang.ClassCastException` is produced.

Software AG Common Platform supports loading multiple versions of class instances in the JVM. However, you must ensure that one execution thread consistently uses the same class type. The OSGi `Export-Package` header supports a `uses` directive to provide clarity in such cases.

For example, in the diagram below, *bundle A* imports a package from *bundle B*. One of the imported packages contains a class with a method signature that includes parameters found in *bundle C* version 1.1. At the same time, *bundle A* has dependencies to another version of *bundle C*, version 1.0, which leads to an invalid classloader graph.



Note:

If you are adding jars to a project's classpath through its `\lib` directory, make sure that containing classes are not already exported by another, similarly to the provided example.

A Differences Between WmTomcat and WmAppPlat

The following table compares the WmTomcat package and the WmAppPlat package based on how each package uses key Tomcat-specific components:

Component	WmTomcat	WmAppPlat
Tomcat version	Apache Tomcat 6.5 Note: This is the Tomcat version used in the last WmTomcat implementation, available in Integration Server 9.12.	Apache Tomcat 8.5.35 Note: This is the Tomcat version used in the first WmAppPlat implementation, available from Application Platform 10.3.
URL of deployed resources	<ul style="list-style-type: none"> ■ uses the port number of an Integration Server HTTP listener ■ contains the web directory 	<ul style="list-style-type: none"> ■ uses the port number of a Tomcat connector ■ does not contain the web directory
docbase	Everything under <code>web/</code> is visible by default.	Only what is declared for servlet and filter mapping in the WAR file is accessible.
ROOT	WmTomcat package home page	Software AG Spring Board
buffer restriction	Integration Server does not support incremental buffering of HTTP responses.	Clients are connecting directly to Tomcat HTTP listeners and use the typical Tomcat buffering methods.
shared jars	All files in <code>ServerClassLoader</code> , including <code>common/lib</code> and <code>common/lib/ext</code> .	All files in the <code>org.apache.catalina</code> host bundle classpath graph.
undeployment	Unloading the IS web package removes the WAR application from the container.	You must remove the WAR project from the <code>Software AG_directory / profiles/IS_default/workspace/webapps/</code> directory path.

Component	WmTomcat	WmAppPlat
server configuration path	<i>Integration Server_directory</i> /web/conf/	<i>Software AG_directory</i> /profiles/IS_default/configuration/tomcat/conf/
port configuration path	Integration Server Administrator	<i>Software AG_directory</i> /profiles/IS_default/ configuration/ com.softwareag.platform. config.propsloader/
security-constraints	Automatically bridges to the Integration Server security.	Leverages the Integration Server JAAS-based configuration.
tomcat.base path	<i>Integration Server_directory</i> /instances/ <i>instance_name</i> /web/	<i>Software AG_directory</i> /profiles/IS_default/configuration/tomcat
Tomcat filter support	Filtering happens at a late stage, after WmTomcat sends a request to the embedded Tomcat.	Fully supported.
logging	You must copy the <code>commons-logging.properties</code> file to the directory where the package containing your web application that uses the Jakarta commons-logging API is located.	WmAppPlat bridges logs to Log4j and writes them to the <code>sag-osgi.log</code> and <code>wrapper.logs</code> files for your web application container.
Tomcat Manager Application	Included in WmTomcat.	Not included in WmAppPlat. You can install it separately. For more information about installing the Tomcat Manager Application, see “Administering Web Applications” on page 39.
security realm	The Integration Server realm is used.	By default, the <code>org.apache.catalina.realm.LockoutRealm</code> is used. You can also add a custom <code>AppPlatformRealm</code> . For information about adding the <code>AppPlatformRealm</code> , see <i>Getting Started with the webMethods Application Platform API</i> .

Component	WmTomcat	WmAppPlat
working directory	WmTomcat uses the <i>Integration Server_directory</i> \ instances\instance_name\web\work directory as a working directory and Apache Tomcat administers this directory.	<i>Software AG_directory</i> /profiles/ IS_default/workspace/work/
