



THE  
POWER  
TO KNOW.

# **SAS<sup>®</sup> AppDev Studio<sup>™</sup> 3.4**

## **Eclipse Plug-ins**

### **User's Guide**



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2010. *SAS® AppDev Studio™ 3.4 Eclipse Plug-ins: User's Guide*. Cary, NC: SAS Institute Inc.

**SAS® AppDev Studio™ 3.4 Eclipse Plug-ins: User's Guide**

Copyright © 2010, SAS Institute Inc., Cary, NC, USA

All rights reserved. Produced in the United States of America.

**For a hardcopy book:** No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

**For a Web download or e-book:** Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

**U.S. Government Restricted Rights Notice:** Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227–19 Commercial Computer Software-Restricted Rights (June 1987).

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, April 2010

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at [support.sas.com/publishing](http://support.sas.com/publishing) or call 1-800-727-3228.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

---

## Contents

<b>Chapter 1 • Installing AppDev Studio</b>	<b>1</b>
Installation Prerequisites	1
Installation and Post-Installation of SAS AppDev Studio	3
Accessibility Features of AppDev Studio	5
<b>Chapter 2 • Overview of AppDev Studio 3.4</b>	<b>7</b>
The SAS AppDev Studio 3.4 Eclipse Plug-ins	7
New Features	7
Migrating Applications to AppDev Studio 3.4	8
<b>Chapter 3 • Projects, Profiles, and Templates</b>	<b>9</b>
SAS Web Application Projects	9
Minimize the Number of Open Projects	10
Server Profiles	10
Templates	11
Template Descriptions	11
<b>Chapter 4 • Walk-Through for Web Infrastructure Platform Templates</b>	<b>15</b>
Introduction	15
Part I: Create the Project and the Application Metadata	16
Part II: Add the JDBC TableView Template	20
Part III: Configure the Tomcat Server	23
Part IV: Add a Welcome Page and Run the Application	26
Add a ReportViewer Servlet Template to the Project	28
Add a SAS Stored Process Servlet Template to the Project	31
Input and Output Parameters	34
<b>Chapter 5 • Walk-Through for Data-Driven Project Creation</b>	<b>37</b>
Create an Information Map Fixed Portlet from Data	37
The Portlet Editor	42
<b>Chapter 6 • Template and Testing Details</b>	<b>45</b>
Files Added By the Metadata Creation Template	45
Copying the Application Metadata	47
Files Added by the Stored Process Java Client Template	47
Files Added by the Stored Process Servlet Template	48
Tomcat Configuration Details	49
Deployment and Authentication	50
<b>Chapter 7 • Exporting Projects</b>	<b>53</b>
Exporting Java and SAS Java Projects as a Set of Jars	53
Exporting a SAS Web Application Project as a WAR File	54
Exporting a Project Using a Deployment Descriptor File	55
<b>Chapter 8 • Managing the Jars in a Project</b>	<b>57</b>
The SAS Repository	57
Opening the SAS Repository Properties Editor	58
Identifying Dependent Jars	59
Removing Jars from the Classpath	59
Changing the Order of Jars in the Classpath	60
Adding New Jars to the Classpath	61

Adding Dependent Jars . . . . .	61
Specifying the Current Versions of Jars . . . . .	62
Specifying Other Jar Versions . . . . .	62
Removing Version Restrictions . . . . .	63
Reporting the Classpath Jars . . . . .	63
Reporting Jar Relationships . . . . .	64
Finding a Class in a Jar . . . . .	65
Changing Default Classes for SAS Java Projects . . . . .	66
<b>Chapter 9 • Using the SAS Editor Extensions . . . . .</b>	<b>67</b>
Introduction to SAS Editor Extensions . . . . .	67
Accessing SAS Component API Documentation . . . . .	68
Adding Missing Import Statements . . . . .	69
Attaching a SAS Model to a Viewer . . . . .	70
SAS Snippets . . . . .	72
<b>Appendix 1 • Creating a SAS Web Application that Does Not Use the Web     Infrastructure Platform . . . . .</b>	<b>75</b>
<b>Index . . . . .</b>	<b>79</b>

## Chapter 1

# Installing AppDev Studio

---

<b>Installation Prerequisites</b> .....	<b>1</b>
Install Eclipse 3.4 .....	1
Upgrade to Web Tools Platform (WTP) version 3.0.5 .....	2
Supported Versions of SAS Software .....	2
Java Platform Requirements .....	2
Running AppDev Studio Releases 3.4 and 3.3 on the Same Machine .....	3
Changes to the SAS AppDev Studio Product Bundle .....	3
<b>Installation and Post-Installation of SAS AppDev Studio</b> .....	<b>3</b>
Installation Instructions .....	3
Post-Installation Configuration .....	4
Eclipse Memory Settings .....	4
Windows Vista Settings .....	5
<b>Accessibility Features of AppDev Studio</b> .....	<b>5</b>

---

## Installation Prerequisites

### *Install Eclipse 3.4*

You must have Eclipse 3.4.2 installed before installing AppDev Studio 3.4. The AppDev Studio installation updates the Eclipse environment, and if you do not have a compatible version of Eclipse installed, you cannot install AppDev Studio. Although Eclipse 3.5 has been released and might function with SAS AppDev Studio 3.4, it has not been tested, and is not supported.

Install Eclipse by following these steps:

1. Create a working directory for Eclipse 3.4 (for example, **C:\Eclipse34**).
2. Download Eclipse 3.4. The supported release of Eclipse can be found on the SAS Third-party Downloads page located at <http://support.sas.com/resources/thirdpartysupport/v92/othersw.html#eclipse>.
3. Extract the Eclipse archive to **C:\Eclipse34**. You should now have an eclipse directory inside your working directory (for example, **C:\Eclipse34\eclipse**).

## Upgrade to Web Tools Platform (WTP) version 3.0.5

SAS Web Application Projects use facilities provided by the Eclipse Web Tools Platform (WTP). For more information about WTP, see <http://www.eclipse.org/webtools>.

If you are using the supported Eclipse 3.4.2, and not a later version of Eclipse, upgrade to WTP 3.0.5. Do not install WTP 3.1, which requires Eclipse 3.5.x, and is not compatible with Eclipse 3.4.x.

To update to WTP 3.0.5, follow these steps:

1. Go to the Web Tools Platform version 3.0.5 archive page: <http://archive.eclipse.org/webtools/downloads/drops/R3.0/R-3.0.5-20090521045405/>.
2. From the **Web Tools Platform** section, download the zip file for Web App Developers (the “wtp” link).
3. Ensure that the Eclipse is closed.
4. Unzip the **wtp-R-3.0.5-20090521045405.zip** file to `\eclipse\dropins\`, preserving the directory structure of the zip file.

The unzip should result in a new eclipse directory inside `\dropins`. For example, `C:\eclipse\dropins\eclipse`.

5. Confirm that WTP 3.0.5 was installed by starting Eclipse, selecting **Help** ⇒ **Software Updates**, and checking the Installed Software list for version 3.0.5 of both the “Web Developer Tools” and the “Java EE Developer Tools.”

## Supported Versions of SAS Software

SAS AppDev Studio 3.4 requires SAS 9.2 software. To develop for SAS 9.1.3SP4 software, you must use SAS AppDev Studio 3.3.

To work correctly, a SAS AppDev Studio 3.4 project must match the maintenance level of the SAS 9.2 installation on which it will be run. You only need to apply maintenance to SAS AppDev Studio if you are developing SAS projects for SAS 9.2 installations that have also had maintenance applied.

## Java Platform Requirements

The SAS AppDev Studio 3.4 development bundle requires Java 2 Standard Edition (J2SE) 1.5.0\_12 or higher for both application development and execution of applications at run time. For execution of web applications, SAS AppDev Studio 3.4 supports the Web browsers, application servers, and Java runtimes as defined in the Third Party Software for SAS 9.2 Foundation document (<http://support.sas.com/resources/thirdpartysupport/v92/index.html>).

In addition to the system requirements listed above, SAS AppDev Studio 3.4 Eclipse Plugins also requires the minimum Java release required by Eclipse as documented in <http://www.eclipse.org/downloads/moreinfo/jre.php>. Specifically, Eclipse IDE for Java EE Developers version 3.4 requires Java 5.

### **Running AppDev Studio Releases 3.4 and 3.3 on the Same Machine**

SAS AppDev Studio 3.4 can be installed on the same machine as AppDev Studio 3.3. However, when using the two releases on the same machine, you should specify different project workspaces. Running SAS AppDev Studio 3.4 on a machine that has a version of AppDev Studio older than 3.3 is not supported.

### **Changes to the SAS AppDev Studio Product Bundle**

Because SAS software release 8.2 is no longer supported by SAS AppDev Studio, the SAS AppDev Studio Server Side Catalogs are no longer included in the SAS AppDev Studio product bundle. These catalogs, provided to support connections to data objects based on SCL on the SAS 8.2 server, have been deprecated in favor of using more standard and scalable data connection technologies such as JDBC (for relational data) and the SAS 9 Java OLAP interfaces (for OLAP data).

---

## **Installation and Post-Installation of SAS AppDev Studio**

### **Installation Instructions**

The SAS Deployment Wizard provides two ways to install AppDev Studio.

The first is to install all the software that you need for development on a single machine. This plan installs and runs all three tiers on the same machine. This includes a scaled down Business Intelligence (BI) server, a middle tier, and the AppDev Studio Eclipse Plug-ins. This type of installation is needed when the necessary servers, such as an Enterprise Business Intelligence (EBI) environment, are not available elsewhere. Running all three tiers on one machine is computationally intensive, so plan accordingly. With this plan you can also interact with existing remote tiers, which AppDev Studio makes easy using profiles. To select this type of installation, use the installation plan **AppDev Studio, one machine, JBoss**.

The second way is to install only the AppDev Studio Eclipse Plug-ins. This installation assumes that the necessary servers already exist. To select this type of installation, use the plan **AppDev Studio, three machine, JBoss**, and install only the AppDev Studio Eclipse Plug-ins client.

Follow these steps to begin a guided installation of AppDev Studio:

1. Start the SAS Deployment Wizard, select **Install SAS Software**, and click **Next**.
2. Select **Perform a Planned Deployment** and **Install SAS Software**.
  - a. If you are installing all three tiers using the one machine plan, also select **Configure SAS Software**.
  - b. If you are installing only the AppDev Studio Eclipse Plug-ins using the three machine plan, clear the **Configure SAS Software** check box.
3. Click **Next**.
4. Follow the on-screen instructions for installing the software.

## Post-Installation Configuration

Before you start AppDev Studio, you must connect it to an Eclipse installation and then configure it.

1. Connect AppDev Studio to an Eclipse installation by following these steps:
  - a. Start the SAS AppDev Studio Eclipse Configuration Tool (**Start** ⇒ **Programs** ⇒ **SAS** ⇒ **SAS AppDev Studio 3.4** ⇒ **SAS AppDev Studio Eclipse Configuration Tool**).
  - b. In the Configuration Tool select **Eclipse** ⇒ **Search** and specify the top level of an Eclipse installation that you want to connect to AppDev Studio.  
  
The search is recursive, and a search of **C:\Eclipse34\** will find the Eclipse installed at **C:\Eclipse34\eclipse**. If you have multiple Eclipse installations under one directory, point to the containing directory to find all the Eclipse installations under that directory.
  - c. When the search is complete, select from the main Configuration Tool window the Eclipse installation that you want to connect to AppDev Studio, and then select **Eclipse** ⇒ **Connect**. The connection process modifies the Eclipse installation to run AppDev Studio and can take several minutes.
  - d. Exit the AppDev Studio Eclipse Configuration Tool.
2. Perform the New Workspace setup for each Eclipse workspace that you want to use.

When you initially launch AppDev Studio within an empty Eclipse workspace, the New Workspace Setup cheat sheet starts and guides you through the following processes:

- creating compatible Java runtimes
- specifying Eclipse compiler options
- setting the correct server run time for web application development
- creating a BI Server Profile
- creating a connection profile for the BI server
- configuring a Tomcat server for testing

To start this cheat sheet, launch the Eclipse attached to AppDev Studio. If the New Workspace Setup cheat sheet is not automatically displayed, select **Help** ⇒ **Cheat Sheets**, and then expand **SAS AppDev Studio** and choose **New Workspace Setup**.

## Eclipse Memory Settings

Because of a memory intensive Java EE task in the Eclipse Web Tools Platform, if you like to have several SAS Web Application projects open in your workspace, you should specify the maximum heap size to be at least 768 MB. On rare occasions this task can be triggered simultaneously on multiple worker threads in Eclipse. When this happens, Eclipse can run out of memory if the heap is not large enough, causing Eclipse to become unstable.

To change the maximum heap size, modify the **-Xmx** setting in the **eclipse.ini** file for the Eclipse installation to which SAS AppDev Studio is connected. For example:

```
-Xmx768m
```

**See Also**

[“Minimize the Number of Open Projects” on page 10](#)

**Windows Vista Settings**

When installed on Windows Vista, SAS AppDev Studio 3.4 can have trouble interacting with SAS 9.2 BI installations. This difficulty stems from Windows Vista enabling IPv6 by default, while the typical SAS 9.2 BI installation is set to use only IPv4 only. To use IPv4 from SAS AppDev Studio, append the following arguments to the `eclipse.ini` file for every Eclipse installation to which SAS AppDev Studio has been connected:

```
-Djava.net.preferIPv4Stack=true
-Djava.net.preferIPv6Addresses=false
```

---

## Accessibility Features of AppDev Studio

SAS AppDev Studio Eclipse 3.4 Plug-ins includes accessibility and compatibility features that improve the usability of the product for users with disabilities. These features are related to accessibility standards for electronic information technology that were adopted by the U.S. Government under Section 508 of the U.S. Rehabilitation Act of 1973, as amended. However, portions of the AppDev Studio interface are implemented using Java controls, and do not always comply with Section 508 guidelines.

Notable exceptions include:

- In some cases, screen-reading technology is unable to read text, read field labels in the correct order, or read only the text that is currently visible.
- The color of the text in the help window of the AppDev Studio Configuration Tool cannot be changed.
- There are no text equivalents for the menu items in `com.sas.servlet.tbeans.dataexplorer.html.VisualDataExplorer`.
- Several fields on `com.sas.servlet.tbeans.dataexplorer.VisualDataExplorer` lack labels.
- The portlet editor does not scroll vertically via keyboard controls.

If you have questions or concerns about the accessibility of SAS products, send e-mail to [accessibility@sas.com](mailto:accessibility@sas.com).



## Chapter 2

# Overview of AppDev Studio 3.4

---

<b>The SAS AppDev Studio 3.4 Eclipse Plug-ins</b> .....	<b>7</b>
<b>New Features</b> .....	<b>7</b>
<b>Migrating Applications to AppDev Studio 3.4</b> .....	<b>8</b>

---

## The SAS AppDev Studio 3.4 Eclipse Plug-ins

The SAS AppDev Studio 3.4 Eclipse Plug-ins support SAS application developers who use the open source Eclipse IDE or a third-party IDE based on the Eclipse platform. Templates are provided that assist with the development of applications and Web applications, including portlets, SAS Stored Processes, Web-based reporting, and OLAP solutions. AppDev Studio 3.4 also enhances the standard Eclipse Java editor by integrating the SAS component API documentation into the Eclipse Help system.

AppDev Studio 3.4 creates SAS Web applications only for SAS 9.2, and those projects are now automatically configured to work with SAS 9.2 and the SAS Web Infrastructure Platform.

---

## New Features

SAS AppDev Studio 3.4 contains the following new features:

- New SAS BI Server profiles make it easy to target and switch between different SAS BI installations during development.
- A new portlet editor for SAS portlet development. New templates for an Editable portlet, a URL display portlet, and a SAS Stored Process portlet. Support for portlet deployment to a SAS Information Delivery Portal.
- New templates for integrating both console and Web applications with SAS Stored Processes via the Stored Process Service API. Support for both streaming results and output parameters.
- New perspective and views for browsing SAS Metadata and creating both SAS Java Application and SAS Web Application projects from metadata objects for SAS Stored Processes and SAS data.
- Enhanced support for managing JDBC connections in both the DataBean Wizard and the JDBC Web application template.

- New and improved cheat sheets to help configure development workspaces and define Web application servers.

---

## Migrating Applications to AppDev Studio 3.4

Because of the major changes in how SAS 9.2 handles metadata, AppDev Studio 3.4 cannot provide a migration tool to move projects from previous versions (you cannot simply import the old project into AppDev Studio 3.4 and have it converted). Instead, you must create a new project using the appropriate template and then move customizations from the existing project into the new SAS 9.2 project. See the *SAS AppDev Studio 3.4 Eclipse Plug-ins Migration Guide* on the AppDev Studio Developer's Site ([support.sas.com/rnd/appdev/](http://support.sas.com/rnd/appdev/)) for information about migrating to AppDev Studio 3.4.

## Chapter 3

# Projects, Profiles, and Templates

---

<b>SAS Web Application Projects</b> .....	<b>9</b>
<b>Minimize the Number of Open Projects</b> .....	<b>10</b>
<b>Server Profiles</b> .....	<b>10</b>
Introduction .....	10
BI Server Profiles .....	10
Metadata Server Connection Profiles .....	11
<b>Templates</b> .....	<b>11</b>
<b>Template Descriptions</b> .....	<b>11</b>
SAS DataBean .....	11
SAS Information Delivery Portal Portlets .....	11
SAS Foundation Services Support .....	12
SAS Web Application Examples .....	12
SAS Web Infrastructure Platform Support .....	13
SAS Stored Process .....	13

---

## SAS Web Application Projects

SAS AppDev Studio Web application development is as flexible as you need it to be. You can add to a project your code and third-party classes or tag libraries such as Apache Struts or JavaServer Faces. You can also add a jar of Java utility classes to a Web application's `\WEB-INF\lib` directory.

SAS Web Application Projects also support the SAS Java Components and the SAS Web Infrastructure Platform, which includes features such as the Logon Manager, themes, and SAS Platform Services. This support is achieved by adding to the project static content, Eclipse Web Tools Platform facets ([www.eclipse.org/webtools/](http://www.eclipse.org/webtools/)), and jars from the SAS Versioned Jar Repository.

The static content can include configuration information, such as declarations in `web.xml`, other configuration files, and also file resources that are served by the Web application.

Facets are a feature of the Web Tools Platform, and define functionality that can be added to a project. Two facets are added to every SAS Web Application Project: the “SAS Java Components” facet and the “SAS Web Infrastructure Platform” facet. (The “SAS Java Components” facet is the SAS 9.2 equivalent of the “SAS Web Module with WIK” facet.) Facets are versioned, and the 9.2.0.0000 version of the SAS Java Components and SAS

Web Infrastructure Platform facets corresponds to the initial release of SAS 9.2. New facets will be available for maintenance releases of SAS. You can view the facets of a SAS Web Application project by opening the project's Properties and selecting **Project Facets**.

In AppDev Studio 3.4, the SAS Web Application Project wizard always adds both facets to a new SAS Web Application project. Neither facet can be removed from the project. This means the SAS Web Application Project wizard cannot be used to create a SAS Web Application project if you do not want to include the "SAS Web Infrastructure Platform" facet. For a process to create a SAS Web application project with only the "SAS Java Components" facet see ["Creating a SAS Web Application that Does Not Use the Web Infrastructure Platform" on page 75](#).

The SAS Versioned Jar Repository is attached to the project and adds the jars needed to support the two included facets. The jars added to the project by the SAS Versioned Jar Repository are also included in the application's `\WEB-INF\lib\` directory. You can view these included jars by opening the project's SAS Repository. See ["Opening the SAS Repository Properties Editor" on page 58](#).

---

## Minimize the Number of Open Projects

Because of the resources devoted to each open project, you should minimize the number of projects that you have open at one time. The fewer projects that you have open, the more responsive the development environment will be.

To open an existing project, right-click the project in the Project Explorer, and select **Open Project**. Open projects are indicated with an open folder icon.

To close an existing project, right-click the project and select **Close Project**.

---

## Server Profiles

### Introduction

To avoid repeatedly entering host names, port numbers, and other server information when developing SAS Web Applications with the Web Infrastructure Platform, AppDev Studio 3.4 provides server profiles that you can define and then use to connect to a SAS Business Intelligence installation or a SAS Metadata server. Because the profiles provide your connection to required servers, you should create these profiles before starting a project.

### BI Server Profiles

BI Server Profiles automate the connection to the required remote services. You can connect via one BI Server Profile at a time.

To create a new BI Server Profile, use the "Create a SAS BI Server Profile" cheat sheet:

1. From Eclipse, select **Help** ⇒ **Cheat Sheets**.
2. Expand **SAS AppDev Studio**, select **Create a SAS BI Server Profile**, and then click **OK**.

If the "Create a SAS BI Server Profile" cheat sheet has already been run, reset it by right-clicking the cheat sheet name and selecting **Restart all tasks**.

## Metadata Server Connection Profiles

Connection Profiles help automate logging into a SAS Metadata Server by managing the relationship between user credentials and SAS BI Server profile. You can connect via one Connection Profile at a time.

To create a new Connection Profile, use the “Create a Connection Profile” cheat sheet:

1. From Eclipse, select **Help** ⇒ **Cheat Sheets**.
2. Expand **SAS AppDev Studio**, select **Create a Connection Profile**, and then click **OK**.

---

## Templates

The SAS templates consist of code that helps you rapidly develop SAS Web Applications or implement a particular feature. You can add these templates to a project when it is created, or add them to an existing project later.

The SAS Web Application Examples templates provide you with code and resources in various states of completion. The amount of work needed to make the example functional depends on the template. Some templates create examples that are ready to run using only the information provided when you add the template.

Although this User's Guide focuses on SAS Web Applications created in AppDev Studio 3.4 or migrated from a previous version, non-Web application projects (SAS Java Projects and Eclipse Web Tools Dynamic Web projects) are also supported in AppDev Studio 3.4.

If you add a SAS Web Application Examples template to an Eclipse Dynamic Web project, the project is converted to a SAS Web Application Project.

---

## Template Descriptions

The SAS AppDev Studio 3.4 templates by category.

### **SAS DataBean**

SAS JDBC Databean Class

creates a JDBC Java data class that provides access to a data table.

### **SAS Information Delivery Portal Portlets**

DisplayURL Portlet

displays the contents of an HTML page. The URL of the page is specified at run time.

Editable Portlet

displays the contents of a string. The string is specified at run time.

Information Map Fixed Portlet

displays the contents of an Information Map. The Information Map is specified at design time and displays the map at run time.

Information Map Runtime Portlet

displays a list of Information Maps defined on your server and displays the selected map.

JSP Portlet

displays the output of a single JSP that is contained in your project.

Remote Portlet

displays the contents of a remote page inside a frame. The URL of the remote page is specified at design time.

Stored Process List Portlet

displays a list of SAS Stored Processes and executes the selected stored processes. The list is specified at design time.

Stored Process Single Portlet

displays the results of executing a single SAS Stored Process. The stored process is specified at design time.

## ***SAS Foundation Services Support***

Context Listener For Default Services

creates a default ServletContextListener class for deploying and destroying default SAS Foundation Services.

Context Listener For Local Services

creates a ServletContextListener class for deploying and destroying local SAS Foundation Services.

Context Listener For Remote Services

creates a ServletContextListener class for deploying and destroying remote SAS Foundation Services.

Context Listener For Remote and Local Services

creates a ServletContextListener class for deploying and destroying local and remote SAS Foundation Services.

JAAS Login Configuration File

creates a login configuration file for JAAS authentication using a SAS Metadata Server.

Logging Configuration

adds a SAS Foundation Services logging configuration file to the Web application.

## ***SAS Web Application Examples***

All the SAS Web Application Examples templates, listed below, use the SAS Web Infrastructure Platform (SAS WIP). For each template listed there is a corresponding template available in the AppDev Studio interface that uses the legacy SAS Foundation Services (SAS FS).

Information Map Default Servlet (uses SAS WIP)

creates a Default Information Map based on the Model 2 (MVC) Web Application Architecture. Both a JSP page and a Java file containing the servlet's class are created.

Information Map OLAPTableView Servlet (uses SAS WIP)

creates an Information Map OLAPTableView example based on the Model 2 (MVC) Web Application Architecture for use with OLAP data. Both a JSP page containing OLAPTableView custom tags and a Java file containing the servlet's class are created.

Information Map TableView Servlet (uses SAS WIP)

creates an Information Map TableView example based on the Model 2 (MVC) Web Application Architecture for use with relational data. Both a JSP page containing TableView custom tags and a Java file containing the servlet's class are created.

Information Map Viewer Servlet (uses SAS Visual Data Explorer and SAS WIP)

creates an Information Map viewer example that uses a servlet to access relational and OLAP data. The selected map is viewed with the Visual Data Explorer. A Java file containing the servlet's class is created.

JDBC Default Servlet (uses SAS WIP)

creates a Default JDBC example based on the Model 2 (MVC) Web Application Architecture. Both a JSP page and a Java file containing the servlet's class are created.

JDBC TableView Servlet (uses SAS WIP)

creates a JDBC TableView example based on the Model 2 (MVC) Web application architecture. Both a JSP page containing TableView custom tags and a Java file containing the servlet's class are created.

Report Viewer Servlet (uses SAS Web Report Viewer and SAS WIP)

creates a report viewer example that uses a servlet to access relational and OLAP data. The selected report is viewed in SAS Web Report Viewer. A Java file containing the servlet's class is created.

SAS Stored Process Servlet (uses SAS WIP)

creates a stored process example that uses a servlet to display the output of a stored process.

## ***SAS Web Infrastructure Platform Support***

Examples Welcome Page

adds a JSP page that lists the SAS Web Applications Examples added to the project.

SAS Web Infrastructure Platform Applications Metadata Creation

creates support files for creating and deploying Application metadata that is required by the SAS Web Infrastructure Platform.

## ***SAS Stored Process***

Java Client for executing a SAS Stored Process

creates a simple Java client that executes a SAS Stored Process and writes the results to a file.



## Chapter 4

# Walk-Through for Web Infrastructure Platform Templates

---

<b>Introduction</b>	<b>15</b>
<b>Part I: Create the Project and the Application Metadata</b>	<b>16</b>
Create a SAS Web Application Project	16
Add the Application Metadata Creation Template	18
Run the Launch File and Create the Application Metadata	19
<b>Part II: Add the JDBC TableView Template</b>	<b>20</b>
<b>Part III: Configure the Tomcat Server</b>	<b>23</b>
<b>Part IV: Add a Welcome Page and Run the Application</b>	<b>26</b>
Add a Welcome Page Template	26
Run the Application	26
<b>Add a ReportViewer Servlet Template to the Project</b>	<b>28</b>
Add the ReportViewer Servlet Template	28
Restart the Server and Run the Application	29
<b>Add a SAS Stored Process Servlet Template to the Project</b>	<b>31</b>
Add the SAS Stored Process Servlet Template	31
Replace a Value in the Servlet Code	32
Restart the Server and Run the Application	32
Change a Stored Process Input Parameter	33
<b>Input and Output Parameters</b>	<b>34</b>
Input Parameters	34
Output Parameters	35

---

## Introduction

The following walk-through guides you through creating and running three of the available Web Infrastructure Platform templates. Many of the templates are similar, and a knowledge of one template will transfer to the others. On-screen help is available for every part of every template. You should complete the entire walk-through to become familiar with the AppDev Studio interface and the template requirements.

This walk-through creates a SAS Web Application Project and then builds upon itself, incrementally adding three templates to the project. After the project is created and the first template added, the Tomcat server is configured to run the application. The second and third templates are added to the project later, but because they use the same server, no changes to the server parameters are necessary.

This walk-through assumes the following:

- a Tomcat Web server was installed and configured according to the cheat sheet “Configure a Tomcat Server for Testing” (part of the “New Workspace Setup” cheat sheet)
- a SAS BI Server Profile exists for the servers that you are developing against
- a Metadata Server Connection Profile exists for the server that you are developing against
- a data table, a Web Report Studio report, and a stored process are available on the SAS Metadata Server. You can use any data source of the appropriate type when you build the example, but this walk-through uses the `sashelp.class` table, a report based on that table, and the stored process “Sample: Multiple Output Formats.”

This walk-through adds the following templates to the project in this order:

1. JDBC TableView Servlet (uses SAS WIP)
2. ReportViewer Servlet (uses SAS Web Report Viewer and SAS WIP)
3. SAS Stored Process Servlet (uses SAS WIP)

---

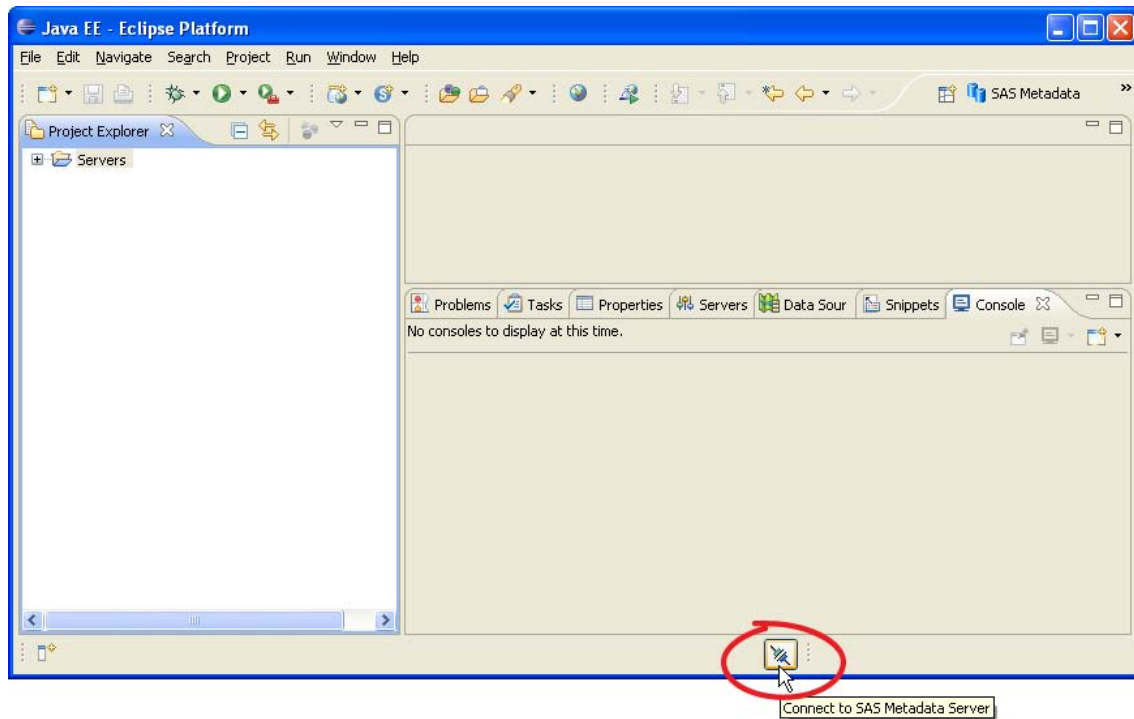
## Part I: Create the Project and the Application Metadata

### **Create a SAS Web Application Project**

Create the project to which you will add the templates.

1. Connect to a SAS Metadata Server.

Although not necessary to create a new project, connecting to a metadata server as the first step in this walk-through ensures that you have defined a BI Server Profile and Metadata Server Connection Profile. For help setting up the profiles, see [“Server Profiles” on page 10](#)



2. Select **File** ⇒ **New** ⇒ **Other**.
3. Expand **SAS AppDev Studio**.
4. Select **SAS Web Application Project**, and click **Next**.
5. For the project name, enter **MyProject**, and click **Next**. In the next section you add the Metadata Creation template.

The project name is used as the context name, and cannot contain spaces.

### Add the Application Metadata Creation Template

The SAS Web Infrastructure Application Metadata Creation template adds to a project the files that enable you to create, delete, and copy the metadata needed to communicate with the Web Infrastructure Platform Logon Manager. Because integration with the SAS Web Infrastructure Platform requires this application metadata, this template should be the first that you add to a project.

1. Select the **Add Template Content** check box.
2. Expand the following folders:
  - **SAS Java Web Application**
  - **SAS Web Application Examples**
  - **SAS Web Infrastructure Platform Support**
3. Select **SAS Web Infrastructure Platform Application Metadata Creation**, and click **Next**.
4. Select the **BI Server Profile** that matches the BI installation that you plan to develop against.

The BI Server Profile that is selected by default is the one associated with the metadata connection profile that you used to connect to the metadata server.

5. Clear the **Application ID** field.

The Application ID should be blank unless there is a specific need to find the application metadata using an ID.

- For the **Port**, enter **8081**. Later, you will configure the Tomcat server to use this port number.

**New Content from a Template**

**SAS Web Infrastructure Platform Application Information**  
Specify application information for the SAS Web Infrastructure Platform

BI Server Profile:

**Application Information**

Application Name:

Description:

Application ID:

**Connection Information**

Protocol:

Web Server Host:  Port:

- Click **Finish**, and then close the **Application.xml** and **LaunchParameters.txt** files.

Although the project is created and the Create Metadata template is added to the project, you must still create the application metadata needed to communicate with the SAS Web Infrastructure Platform.

### **Run the Launch File and Create the Application Metadata**

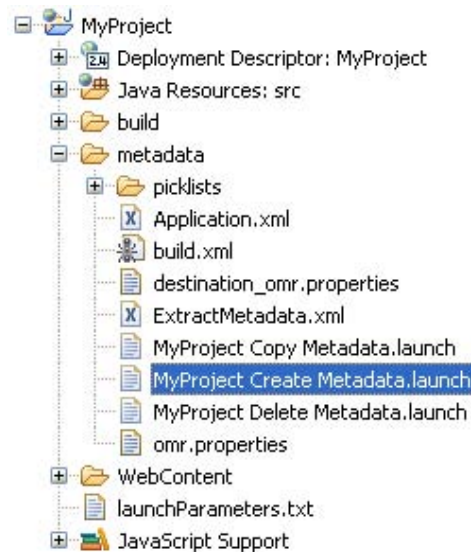
When you added the Create Metadata template, an Ant launch configuration file named **MyProject Create Metadata.launch** was added to the project. The application metadata identifying the Web application to the Web Infrastructure Platform Logon Manager is created by executing the launch file.

Metadata files created during the execution of this launch file appear in `\metadata\temp\`. For information about the files added to the project as a result of adding this template and creating the metadata, see [“Files Added By the Metadata Creation Template” on page 45](#).

- Ensure that the SAS Metadata Server is running in the BI installation whose SAS BI Server Profile was selected earlier.
- Select **Window** ⇒ **Show View** ⇒ **Console** to ensure that the Console is visible.
- Expand the project's metadata folder.

4. Right-click the **MyProject Create Metadata** file, and select **Run as** ⇒ **MyProject Create Metadata**.

Verify that BUILD SUCCESSFUL appears at the end of the output logged to the Console.




---

## Part II: Add the JDBC TableView Template

1. Select **File** ⇒ **New** ⇒ **Other**.
2. Expand **SAS AppDev Studio**.
3. Select **Add Template Content to Project**, and click **Next**.
4. Expand **SAS Java Web Application** and **SAS Web Application Examples**.
5. Select **JDBC TableView Servlet (uses SAS WIP)**, and click **Next**.
6. Click **Next** to accept the Template Configuration Parameters.
7. Accept the **BI Server Profile** by clicking **Next**. The BI Server that you plan to develop against should already be selected.
8. Enter the user name and password for the selected server.

**New Content from a Template**

**JDBC Connection Properties**

Select a driver and enter appropriate values to define a JDBC connection.

**Driver Information**

JDBC driver: SAS IOM JDBC Driver

Driver class: com.sas.rio.MVADriver

Datasource URL: jdbc:sasiom://BIserver.place.com:8591

Driver Properties

**Server Information**

Host: BIserver.place.com Port: 8591

☐ Automatically update fields and properties with historical values when host or port is modified

**User Information**

User name: UserID

Password: \*\*\*\*\*

Test Connection

< Back Next > Finish Cancel

9. Click **Test Connection**.

If you do not receive the message “Connection test succeeded,” ensure that the BI Server Profile is correct and the BI Server is running on the expected port.

In the future, when you want to provide a path to your data, click **Driver Properties** and define the **librefs** property.

Because this template is being added to a SAS Web Application project that integrates with the SAS Web Infrastructure Platform, the user credentials entered here are not used when the Web application is run. Instead, the code generated by the template uses the credentials associated with the user that is logged on to the Web application.

10. Click **Next**.
11. For the **Query**, enter `select * from sashelp.class`.
12. Click **Submit Query** to test the query and display the resulting column names.

**New Content from a Template**

**JDBC Query Specification**

Query:

Columns:

Column Name	
Name	
Sex	
Age	
Height	
Weight	

13. Click **Next**.
14. Accept the Servlet Class Parameters by clicking **Next**.

**New Content from a Template**

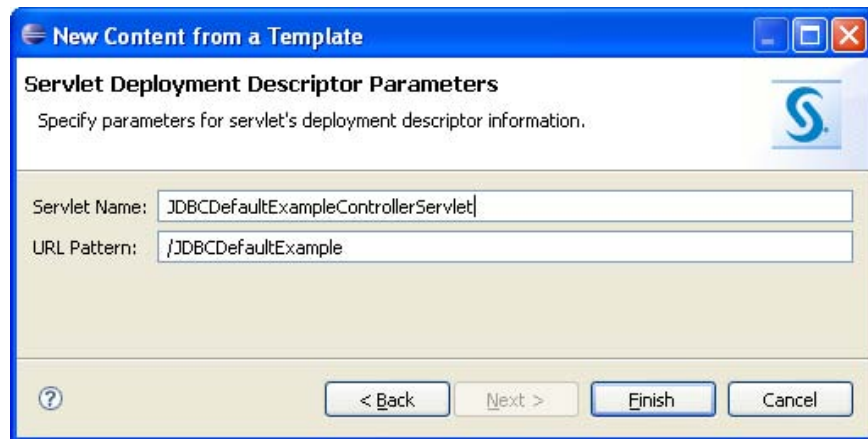
**Servlet Class Parameters**

Specify parameters for creating the servlet class.

Servlet Class Name:

Servlet Package:

15. Click **Finish** to accept the Servlet Deployment Descriptor Parameters and add the JDBC TableView Servlet template to the project.

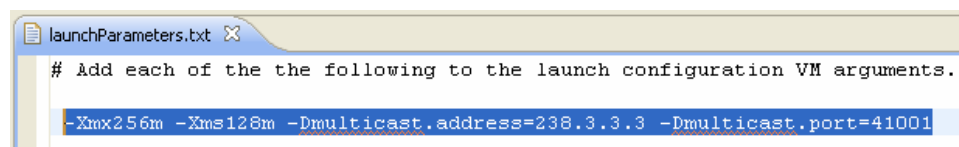


The `launchParameters.txt` file and the JSP and Java files for the servlet are opened. Close all but the `launchParameters.txt` file.

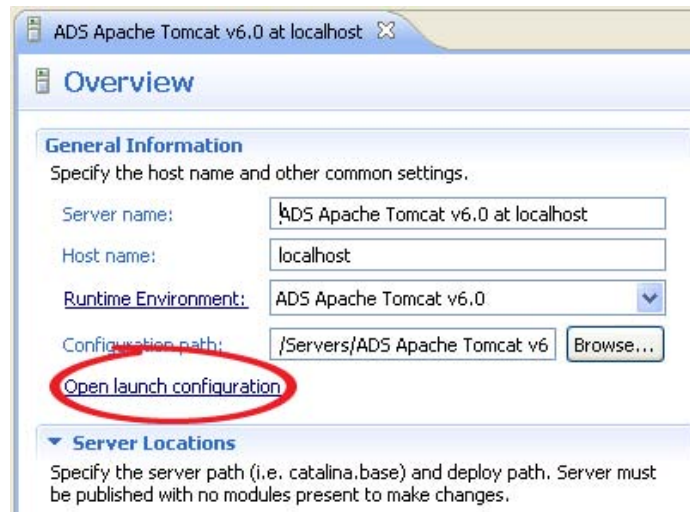
## Part III: Configure the Tomcat Server

You must add launch arguments to the server when testing a SAS Web Application project. These arguments tie the server to a particular SAS 9.2 BI installation whose Remote Services are using those same settings. The added arguments are usually memory settings and the `multicast.address` and `multicast.port`. If the Remote Services are using an authentication token, the `multicast_authentication_token` must also be added. Additional system properties can be added when necessary. For details, see [“Tomcat Configuration Details” on page 49](#).

1. In the `launchParameters.txt` file, edit the parameters so that they are on one line, and then copy that line to the clipboard.

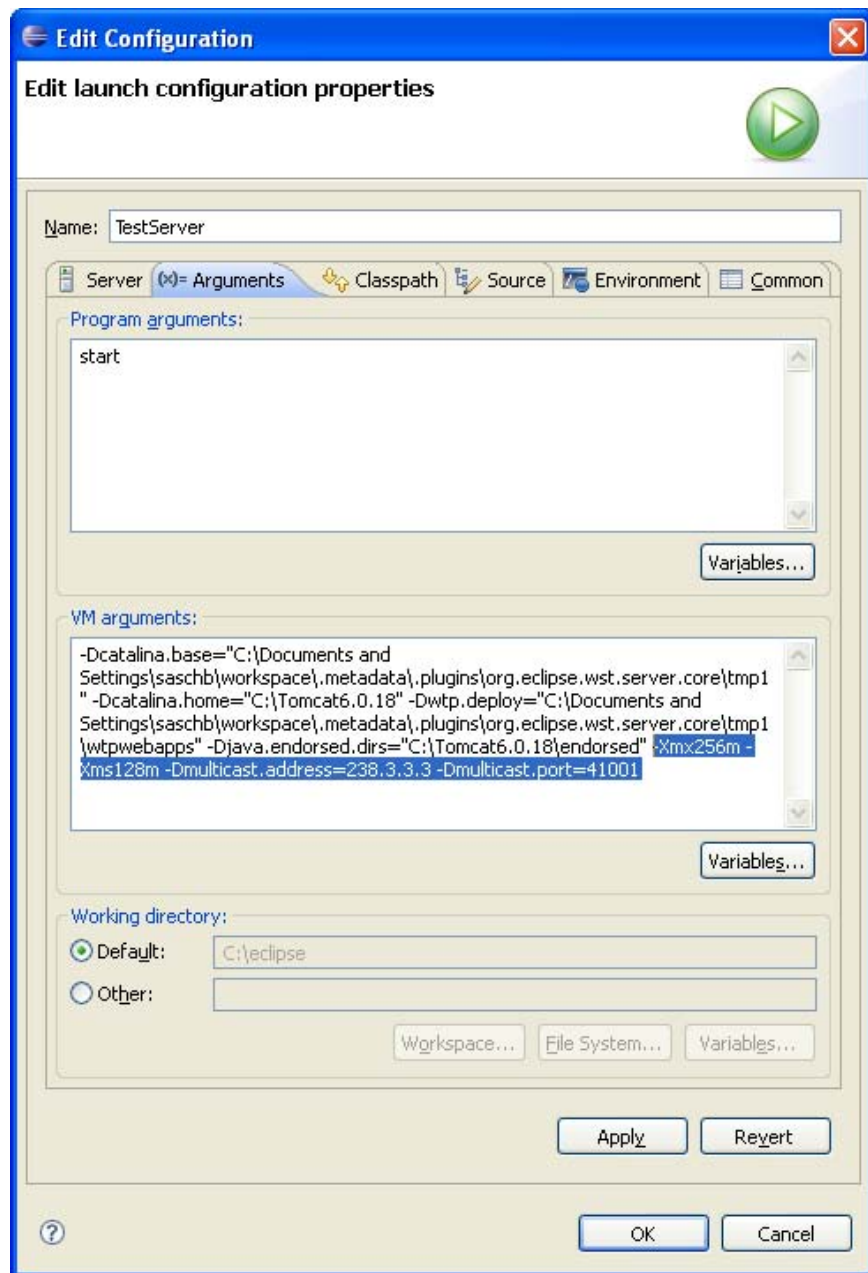


2. From the Servers view (**Window** ⇒ **Show View** ⇒ **Servers**), double-click the Tomcat server to open the server editor.
3. Click the **Open launch configuration** link in the General Information section.



4. In the Edit Configuration dialog box that appears, select the **Arguments** tab.
5. Scroll to the bottom of the **VM arguments** field and place the cursor at the end of the existing contents.
6. Enter a SPACE at the end of the existing content, and then paste the launch parameters from the clipboard.

Remove any arguments that might have been duplicated. In the following display the pasted arguments are highlighted.



7. Click **OK**.
8. On the right side of the server editor, ensure that the HTTP/1.1 port is set to 8081 (the port that you used when adding the metadata creation template).

Ports	
Modify the server ports.	
Port Name	Port Number
Tomcat admin port	8006
HTTP/1.1	8081
AJP/1.3	8010

9. Close the Tomcat server editor and the **launchParameters.txt** file.

The server is now configured and ready to test the application.

## Part IV: Add a Welcome Page and Run the Application

### Add a Welcome Page Template

To help make testing as easy as possible, the Welcome Page template defines a starting point in the Web application that is suitable for integration with the SAS Web Infrastructure Platform. The Welcome Page template works in conjunction with the Web application example templates that add data about the example to `\WEB-INF`

`\sas_examples.xml`.

The Welcome Page template adds a JSP file (default name `sas_examples.jsp`) to the project, and appends the filename to the welcome-file-list declaration in `\WebContent\WEB-INF\web.xml`.

At run time, after successfully logging on to the SAS Web Infrastructure Platform Logon Manager, this JSP is displayed provided that it is the first resource in the welcome-file-list that exists in the Web application. When the Welcome Page is displayed, a link is available for each SAS Web application template added to the project.

If other resources in the welcome-file-list exist in the Web application, you might need to move the `sas_examples.jsp` entry to the top of the list to ensure that it is executed immediately after logon. Because the Examples Welcome Page is not a production entry point, you can move it to the top of the list when you want to use its features and move it down or remove it when you want a different welcome file served.

Add a Welcome Page template by following these steps:

1. From Eclipse, select **File** ⇒ **New** ⇒ **Other**
2. Select **Add Template Content to Project**, and click **Next**.
3. For **Project**, select MyProject.
4. Expand **SAS Java Web Application** and then **SAS Web Infrastructure Platform Support**.
5. Select **Examples Welcome Page**, and then click **Next**, and then **Finish**.

### Run the Application

1. From the Servers View, right-click the server and select **Add and Remove Projects**.
2. Select **MyProject** and add it to the list of **Configured projects**.
3. Click **Finish**.
4. Right-click the server and select **Publish**.

The application is copied to the server. Wait until the status of the server and project is **Synchronized** before proceeding. The first time an application is published to the server, all the static content and the jars are copied. This content is over 100 megabytes, and the time it takes to copy it can exceed the server startup time-out, resulting in a failure to start the server if you attempt to start the server before the copying is complete. Subsequent publishes only copy files that have changed.

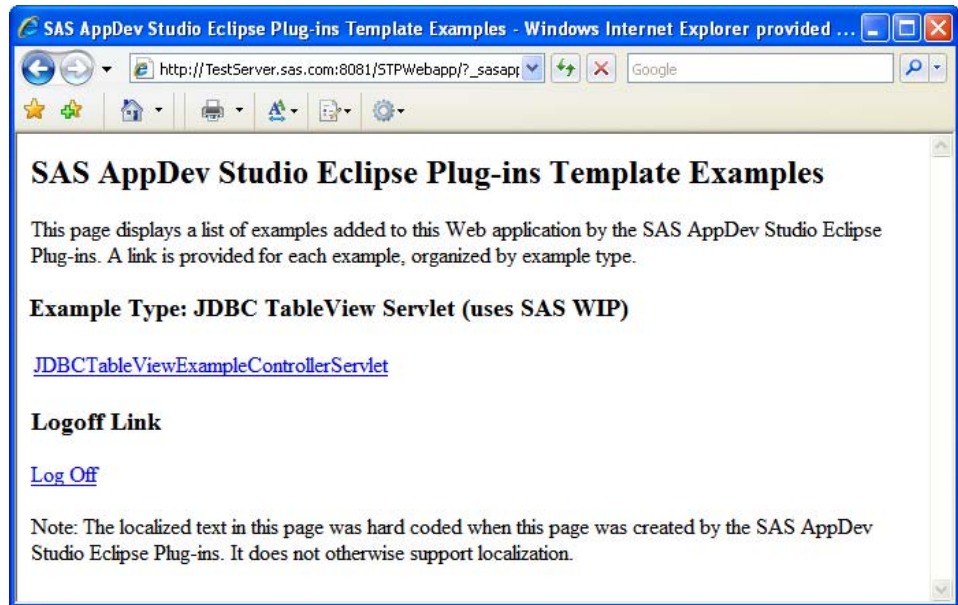
5. From the Servers View, Start the server. Wait until the server State is **Started**.



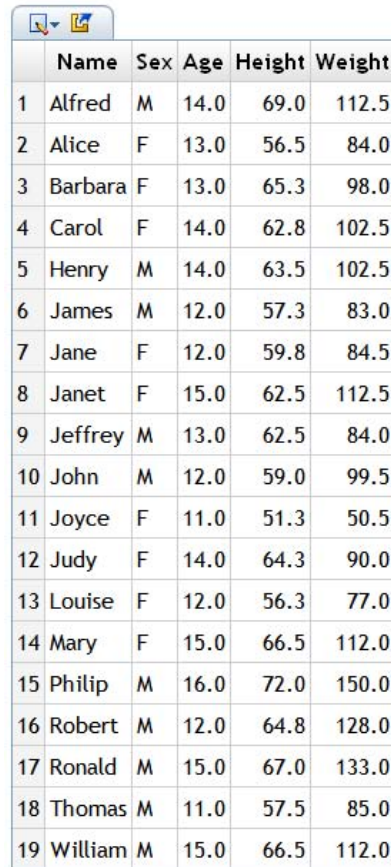
6. In the Project Explorer, right-click MyProject and select **Run As** ⇒ **Run on Server**.
7. Ensure that the correct server is selected and click **Finish**.

Because the server is already started, the Web browser should open to the BI Server page.

8. Log on to the BI Server.
9. The Welcome Page is displayed.



10. Click the link for the JDBC TableView Example.



	Name	Sex	Age	Height	Weight
1	Alfred	M	14.0	69.0	112.5
2	Alice	F	13.0	56.5	84.0
3	Barbara	F	13.0	65.3	98.0
4	Carol	F	14.0	62.8	102.5
5	Henry	M	14.0	63.5	102.5
6	James	M	12.0	57.3	83.0
7	Jane	F	12.0	59.8	84.5
8	Janet	F	15.0	62.5	112.5
9	Jeffrey	M	13.0	62.5	84.0
10	John	M	12.0	59.0	99.5
11	Joyce	F	11.0	51.3	50.5
12	Judy	F	14.0	64.3	90.0
13	Louise	F	12.0	56.3	77.0
14	Mary	F	15.0	66.5	112.0
15	Philip	M	16.0	72.0	150.0
16	Robert	M	12.0	64.8	128.0
17	Ronald	M	15.0	67.0	133.0
18	Thomas	M	11.0	57.5	85.0
19	William	M	15.0	66.5	112.0

11. Close the JDBC TableViewer window and log off.

You should always log off when you are finished testing. Stopping the test server while still logged in might not completely remove the session.

---

## Add a ReportViewer Servlet Template to the Project

### Add the ReportViewer Servlet Template

This section adds a template to an existing project. Because the template uses the same Tomcat server, server parameters, and Welcome Page to run the ReportViewer Servlet, you need to provide only the information collected when adding the template.

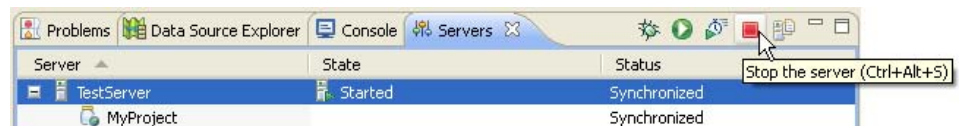
1. Select **File** ⇒ **New** ⇒ **Other**.
2. Expand **SAS Appdev Studio**, select **Add Template Content to Project**, and click **Next**.
3. Expand **SAS Java Web Application** and **SAS Web Application Examples**.
4. Select **Report Viewer Servlet (uses SAS Web Report Viewer and SAS WIP)**, and click **Next**.
5. Accept the Template Configuration Parameters by clicking **Next**.
6. Accept the SAS Web Infrastructure Platform Information by clicking **Next**. The proper BI Server should already be selected.

7. Accept the Servlet Class Parameters by clicking **Next**.
8. Accept the Servlet Deployment Descriptor Parameters by clicking **Finish**.
9. Close the three opened files.

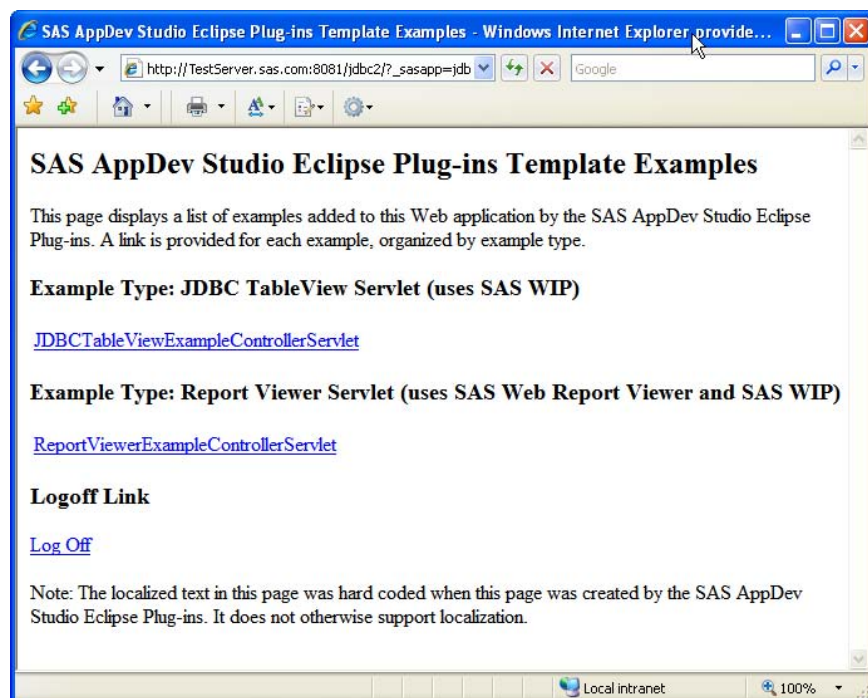
### **Restart the Server and Run the Application**

1. From the Servers View, stop the server (do not use Restart). Wait until the server State is **Stopped**.

For why you should avoid using the Restart command, see the “[Tomcat Shutdown Issue](#)” on page 50.



2. From the Servers View, start the server.
3. In the Project Explorer, right-click MyProject and select **Run As** ⇒ **Run on Server**.
4. Ensure that the correct server is selected and click **Finish**.
5. Log on to the BI server.
6. The Welcome Page is displayed with links for both added templates: the original JDBC TableView Servlet, and the new ReportViewer Servlet.



7. Click the link for the ReportViewer Servlet.

Search		
Location: <span>SAS Folders</span> <span>Up one level</span> <span>Show description</span>		
Name	Type	Date Modified
My Folder	Folder	01/22/2010
Products	Folder	11/30/2009
Shared Data	Folder	11/30/2009
System	Folder	11/30/2009
Users	Folder	11/30/2009

8. Navigate to and click on a report to display it.

Search		
Location: <span>Shared Data</span> <span>Up one level</span> <span>Show description</span>		
Name	Type	Date Modified
SASApp – OLAP Schema	Folder	11/30/2009
ClassSharedData.srx	SAS report	01/25/2010

Report

SAS Web Report Studio : View Report - Windows Internet Explorer provided by SAS

http://BIserver.place.com:8080/SASWebReport/

Back to RFS Log Off SAS Demo User | Preferences | Help

SAS Web Report Studio • ClassSharedData

File View Data Edit View 1 / 1

Table of Contents

Section1

(No group breaks are defined.)

Section Data Options

ClassBasic

Age Height Name Sex Weight

Applied filters: None

Age	Height	Name	Sex	Weight
14	69	Alfred	M	112.5
13	56.5	Alice	F	84
13	65.3	Barbara	F	98
14	62.8	Carol	F	102.5
14	63.5	Henry	M	102.5
12	57.3	James	M	83
12	59.8	Jane	F	84.5
15	62.5	Janet	F	112.5
13	62.5	Jeffrey	M	84
12	59	John	M	99.5
11	51.3	Joyce	F	50.5
14	64.3	Judy	F	90
12	56.3	Louise	F	77
15	66.5	Mary	F	112
16	72	Philip	M	150
12	64.8	Robert	M	128
15	67	Ronald	M	133
11	57.5	Thomas	M	85
15	66.5	William	M	112

Local intranet 100%

---

## Add a SAS Stored Process Servlet Template to the Project

### Add the SAS Stored Process Servlet Template

In this section, as you did earlier in the walk-through, you are adding a template to an existing project. The server is already configured for testing.

1. Select **File** ⇒ **New** ⇒ **Other**.
2. Expand **SAS Appdev Studio**, select **Add Template Content to Project**, and click **Next**.
3. Expand **SAS Java Web Application** and then **SAS Web Application Examples**.
4. Select **SAS Stored Process Servlet (uses SAS WIP)**, and click **Next**.
5. Click **Next** to accept the Template Configuration Parameters.
6. Accept the **BI Server Profile** by clicking **Next**. The BI Server that you plan to develop against should already be selected.
7. Select a stored process by clicking **Change**, selecting **Sample: Multiple Output Formats**, and then clicking **OK**.

Click **Next**.

The screenshot shows a Windows-style dialog box titled "New Content from a Template". Inside, the title "SAS Stored Process Servlet" is displayed above the instruction "Enter settings about the SAS Stored Process which will be executed in a servlet." and a small SAS logo. The "Connection profile" is set to "MDconnectionProfile" with a "Change..." button. The "Stored Process" section has a "Name" field containing "Sample: Multiple Output Formats" and a "Change..." button. Below this are two checkboxes: "Generate streaming results in browser" (checked) and "Generate list of output parameter results" (unchecked). The "Output" section has a checkbox for "Write SAS log to ServletContext log" which is unchecked. At the bottom, there are four buttons: a help icon (?), "< Back", "Next >", and "Finish".

8. Click **Next** to accept the Servlet Class Parameters, and then click **Next** again to accept the Servlet Deployment Descriptor Parameters.
9. Click **Finish**.

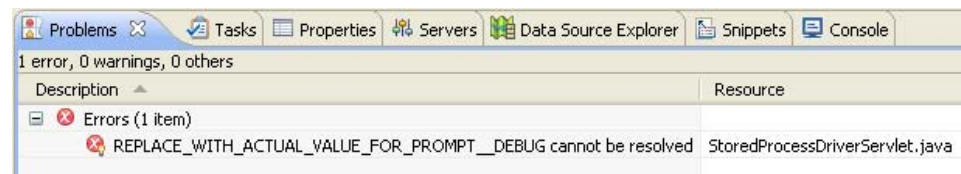
The `StoredProcessWebApp` project is now created and the SAS Stored Process Servlet template is added. Close the `launchParameters.txt` file, but leave open the Java file containing the servlet (`StoredProcessDriverServlet.java`).

## Replace a Value in the Servlet Code

The generated servlet code lacks a value for one of the stored process input parameters. The default value was not defined in the metadata for the stored process, and therefore could not be included in the generated code. You must edit the code and provide a valid value before the servlet can compile and run.

1. Display the Problems view if it is not visible (**Window** ⇒ **Show View** ⇒ **Problems**).
2. Look in the Problems view for this error:  
`REPLACE_WITH_ACTUAL_VALUE_FOR_PROMPT__DEBUG`.

If the error is not in the Problems view, ensure that automatic building is enabled (**Project** ⇒ **Build Automatically**).



3. Double-click the error in the Problems view to go to the error in the Java file.

```
// Prompt: _debug
private static final String _DEBUG_KEY = "_debug";
private static final Object _DEBUG_VALUE = REPLACE_WITH_ACTUAL_VALUE_FOR_PROMPT__DEBUG;
```

4. Replace the problem value with `"log"`, including the quotation marks.

```
// Prompt: _debug
private static final String _DEBUG_KEY = "_debug";
private static final Object _DEBUG_VALUE = "log";
```

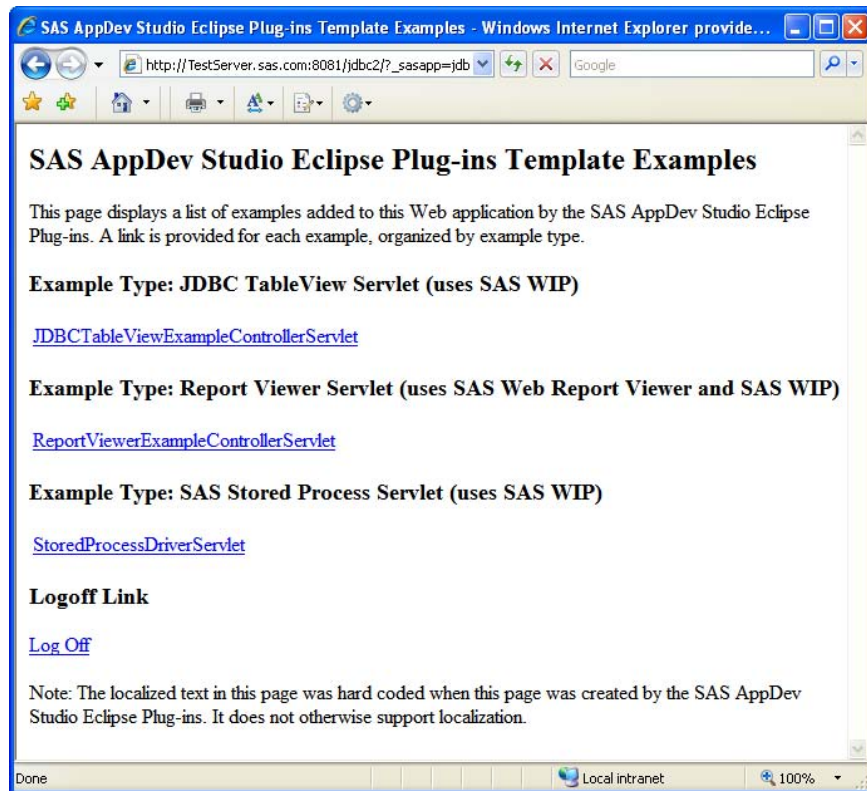
5. Save the file.

The error disappears, assuming you have automatic building enabled.

For more information about input parameters, including how to see what input parameter values and data types are valid in the code, see [“Input and Output Parameters” on page 34](#).

## Restart the Server and Run the Application

1. Stop and then start the server from the Servers View (do not use Restart).
2. Right-click the project and select **Run As** ⇒ **Run on Server**.
3. Ensure that the correct server is selected and click **Finish**.
4. Log in.
5. The Welcome Page is displayed with all three templates listed.



- Click the Stored Process Servlet. The output is HTML, the default for this stored process.

### Data Set SASHELP.RETAIL in HTML Format

Retail sales in millions of \$	DATE	YEAR	MONTH	DAY
\$220	80Q1	1980	1	1
\$257	80Q2	1980	4	1
\$258	80Q3	1980	7	1
\$295	80Q4	1980	10	1

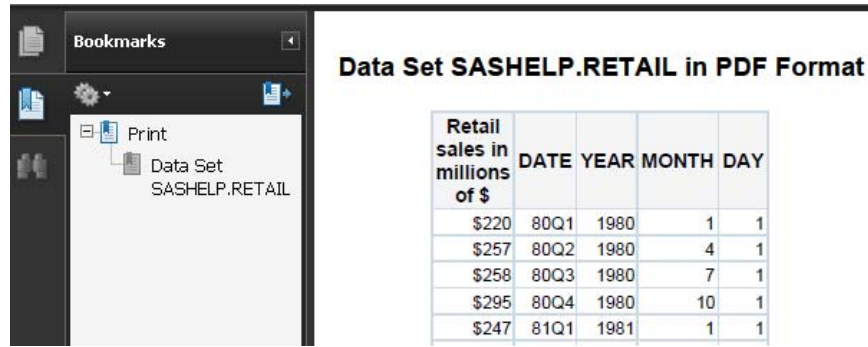
### Change a Stored Process Input Parameter

One of the advantages of using a stored process is that you can use input parameters in the Java code to change the behavior of the servlet. For example, to change the output type of the “Sample: Multiple Output Formats” stored process used in this walk-through to PDF or XML, follow these steps:

- Locate and open `StoredProcessDriverServlet.java`.
- Search for `_ODSDEST_VALUE =`, and then set that variable to **PDF** or **XML**.
- Save the file.
- Stop and then start the server (do not use Restart), and then run the application by right-clicking the project and selecting **Run As** ⇒ **Run on Server**.

For why you should avoid using the Restart command, see the “[Tomcat Shutdown Issue](#)” on page 50.

- On the Welcome Page, click the Stored Process Servlet. The output is a PDF.



You can also change the data set used by the servlet by changing the `DATASET_VALUE`.

Note that the input parameters changed in this walk-through are specific to the “Sample: Multiple Output Formats” stored process. Other stored processes will have different input parameters.

To see how drill-down functionality works, add another SAS Stored Process Servlet template to the project, and use the European Demographic Data stored process.

## Input and Output Parameters

### Input Parameters

Input parameters enable you to pass values to a stored process. The default value of an input parameter, if it is defined, is in the metadata for the stored process.

If a default value is defined for an input parameter, and the data type of the value is text or numeric (integer or floating-point), the input parameter is set to the default value.

If there is no default value defined for an input parameter, or if the type of the input parameter is not text or numeric, then the input parameter is set to `REPLACE_WITH_ACTUAL_VALUE_FOR_PROMPT_YYY`, where `YYY` is the name of the input parameter. This fabricated text causes a compiler error that is listed in the Eclipse Problems view. To build the project, replace the text with a valid value for that input parameter.

Input parameters are represented as prompts in SAS Management Console. To examine the metadata for an input parameter, including its type and default value, follow these steps while using SAS Management Console:

- Right-click the stored process and select **Properties**.
- Change to the **Parameters** tab.
- Select a prompt (input parameter) and click **Edit**.
- Change to the **Prompt Type and Values** tab.

For more information about stored processes and input parameters, see the following resources:

- the comments in the generated code (`StoredProcessDriver.java` or `StoredProcessDriverServlet.java`).

- the *SAS Stored Processes: Developer's Guide* available on the SAS Integration Technologies documentation page at <http://support.sas.com/documentation/onlinedoc/inttech/index.html>, in particular Appendix 3, “Formatting Prompt Values and Generating Macro Variables from Prompts”.
- the SAS Foundation Services Javadoc contained in the SAS BI API Documentation on the SAS AppDev Studio 3.4 Developer's Site at <http://support.sas.com/rnd/appdev/>. In particular, examine the package summary for `com.sas.services.storedprocess`.

## Output Parameters

SAS Stored Processes can return results to a client via output parameters. Output parameters are defined in the metadata for a stored process using SAS Management Console. When a stored process returns output parameters, you can retrieve the values from the `StoredProcessFacade` with the `getOutputParametersWithValues()` method. The values are returned in a `java.util.List`.

An example of retrieving the output parameter values from the `StoredProcessFacade` and writing the results to the output file can be found in the main driver class of the `Stored Process Java Client` template.

An example of retrieving the output parameter values from the `StoredProcessFacade` and writing the results to the browser can be found in the main driver servlet class of the `Stored Process Servlet` template.

In both examples, you can modify the processing of the output parameter results as appropriate to the SAS Stored Process executed in the example.

For a more detailed explanation of output parameters, see “Using Output Parameters” in the *SAS Stored Processes Developer's Guide* available on the SAS Integration Technologies documentation page at <http://support.sas.com/documentation/onlinedoc/inttech/index.html>.



## Chapter 5

# Walk-Through for Data-Driven Project Creation

---

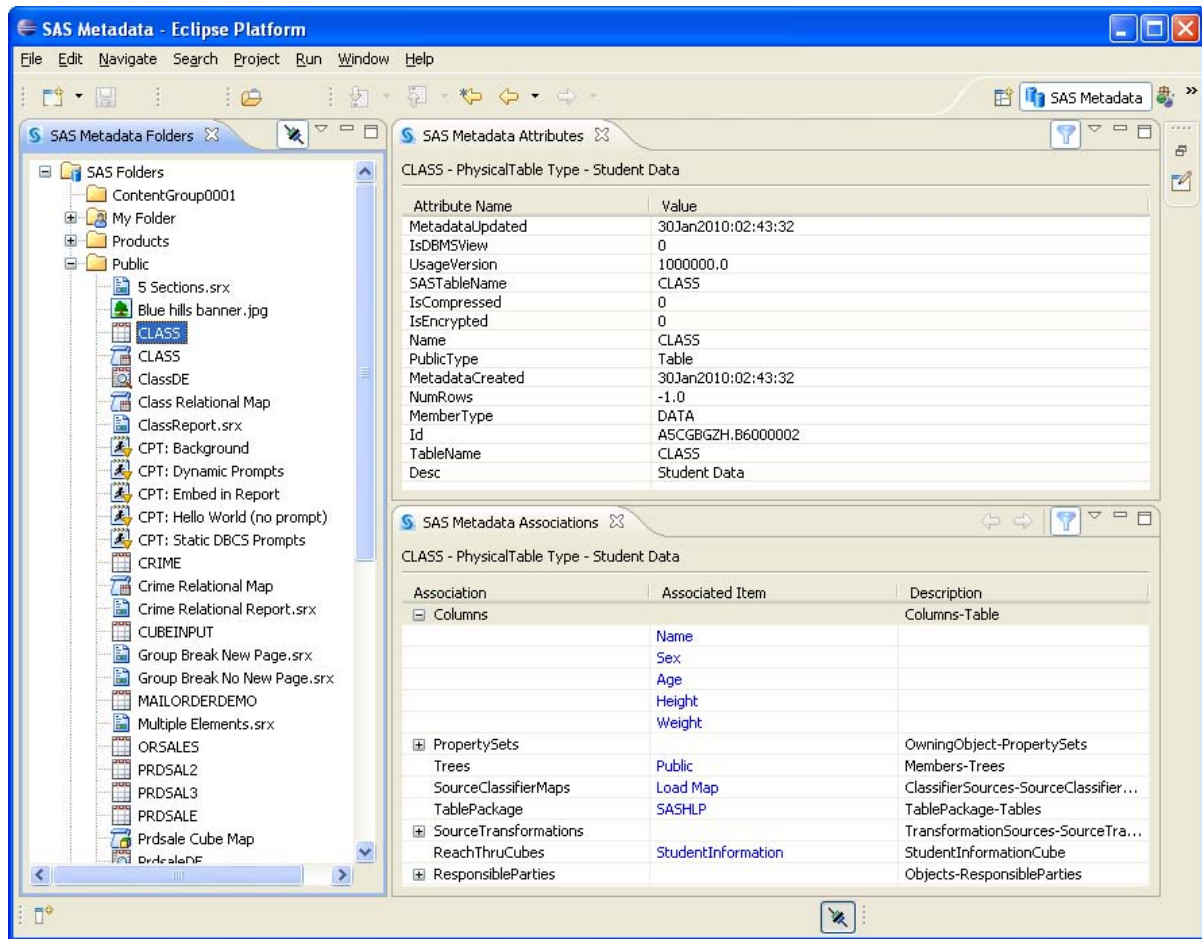
<b>Create an Information Map Fixed Portlet from Data</b> . . . . .	<b>37</b>
Introduction . . . . .	37
Switch to the SAS Metadata Perspective and Create the Project . . . . .	38
Deploy the Portlet . . . . .	40
View the Portlet in the SAS Information Delivery Portal . . . . .	41
<b>The Portlet Editor</b> . . . . .	<b>42</b>

---

## Create an Information Map Fixed Portlet from Data

### *Introduction*

This walk-through explains project creation from the SAS Metadata perspective. The SAS Metadata perspective functions as both a metadata explorer and as a starting point for data-driven development. You can create projects from tables, information maps, and stored processes. From those data sources you can create SAS Web Application projects or SAS Java Projects. The type of template that you can add to a project is determined by the type of data that you select from the SAS Metadata perspective.



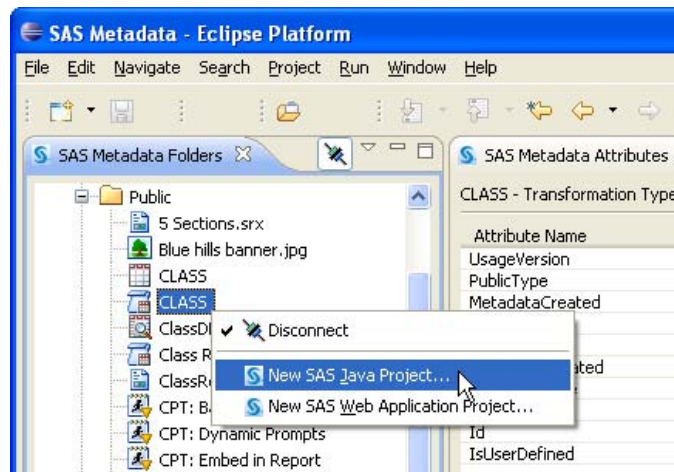
This walk-through assumes the following:

- you have access to a SAS Information Delivery Portal.
- you have access to an Information Map on the Metadata server.
- you have write access to the portal deployment directory. For example, `\Lev1\Web\Applications\SASPortlets4.2\Deployed`.
- a Metadata Server Connection Profile exists for the server that you are developing against.

### Switch to the SAS Metadata Perspective and Create the Project

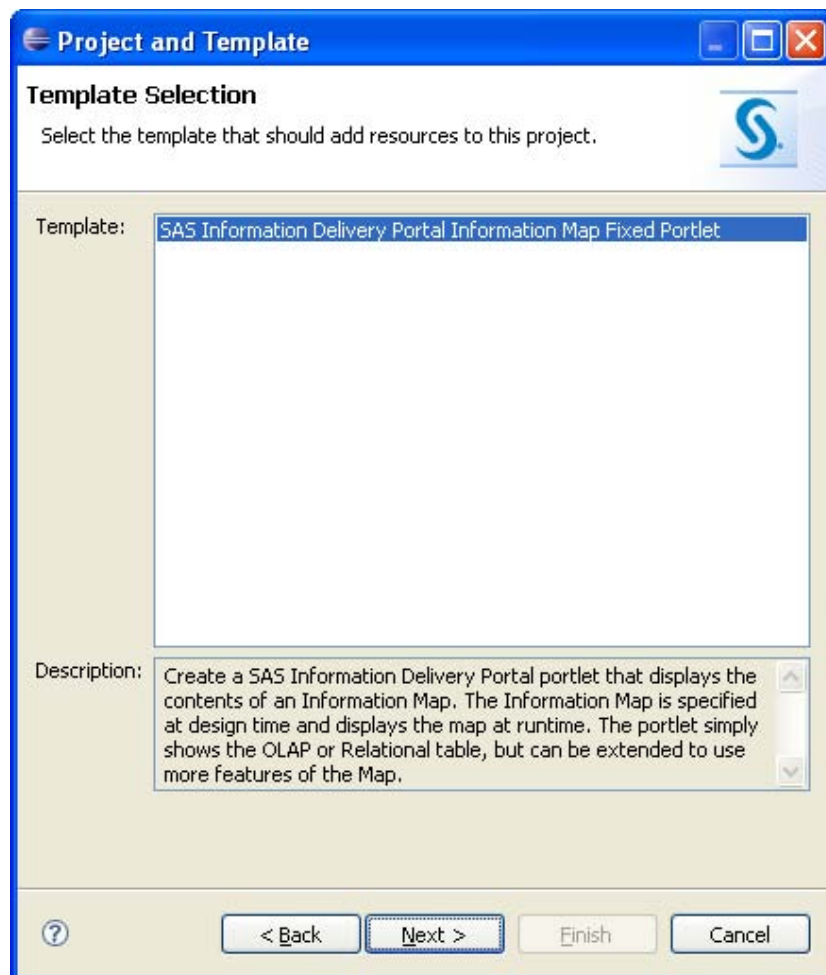
1. Connect to the SAS Metadata Server if you are not already.
2. Switch to the SAS Metadata perspective. Select **Window** ⇒ **Open Perspective** ⇒ **Other**, and then choose **SAS Metadata**.
3. Navigate to an Information Map. Right-click it and select **New SAS Java Project**.

This example uses an Information Map based on the `sashelp.class` data set.



4. Name the project **fixedPortlet**, and click **Next**.
5. Click **Next** to accept the selected template: SAS Information Delivery Portal Information Map Fixed Portlet.

The template list is determined by the type of data that you selected from the Metadata perspective.



6. Click **Finish** to accept the Information Map Fixed Portlet configuration and create the project.

**New Content from a Template**

**Information Map Fixed Portlet**

Enter information below to define the Information Map Fixed Portlet.

Name:

Title:

Description:

Scope:

Package:

Runtime:

**Options**

☐ Pass Context ID

☒ User Can Create More

☒ Auto Deploy

**Deployment**

Directory:

### Deploy the Portlet

To deploy the portlet to the SAS Information Delivery Portal, create a par (portlet archive) file and copy it to the deployment directory. The par file contains all generated classes and static content.

1. Switch to the Java perspective.

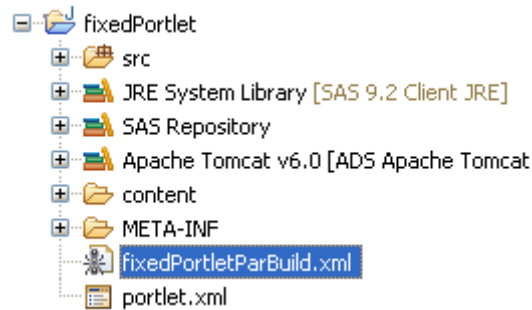


2. Close the `portlet.xml` file.

For information about the `portlet.xml` editor, see “The Portlet Editor” on page 42.

3. Open the project directory, right-click `fixedPortletParBuild.xml`, and select **Build and Copy Par**.

This XML file was created when you added the Fixed Portlet template, and follows the pattern `<projectName>ParBuild.xml`. You can specify additional content to include in the par file by editing this file.

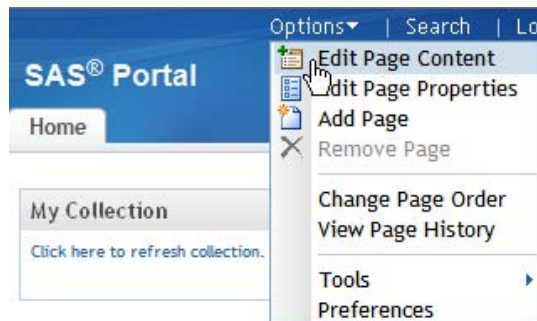


4. Browse to or enter the path for the deployed portal directory: `\Lev1\Web\Applications\SASPortlets4.2\Deployed`, and click **OK**.

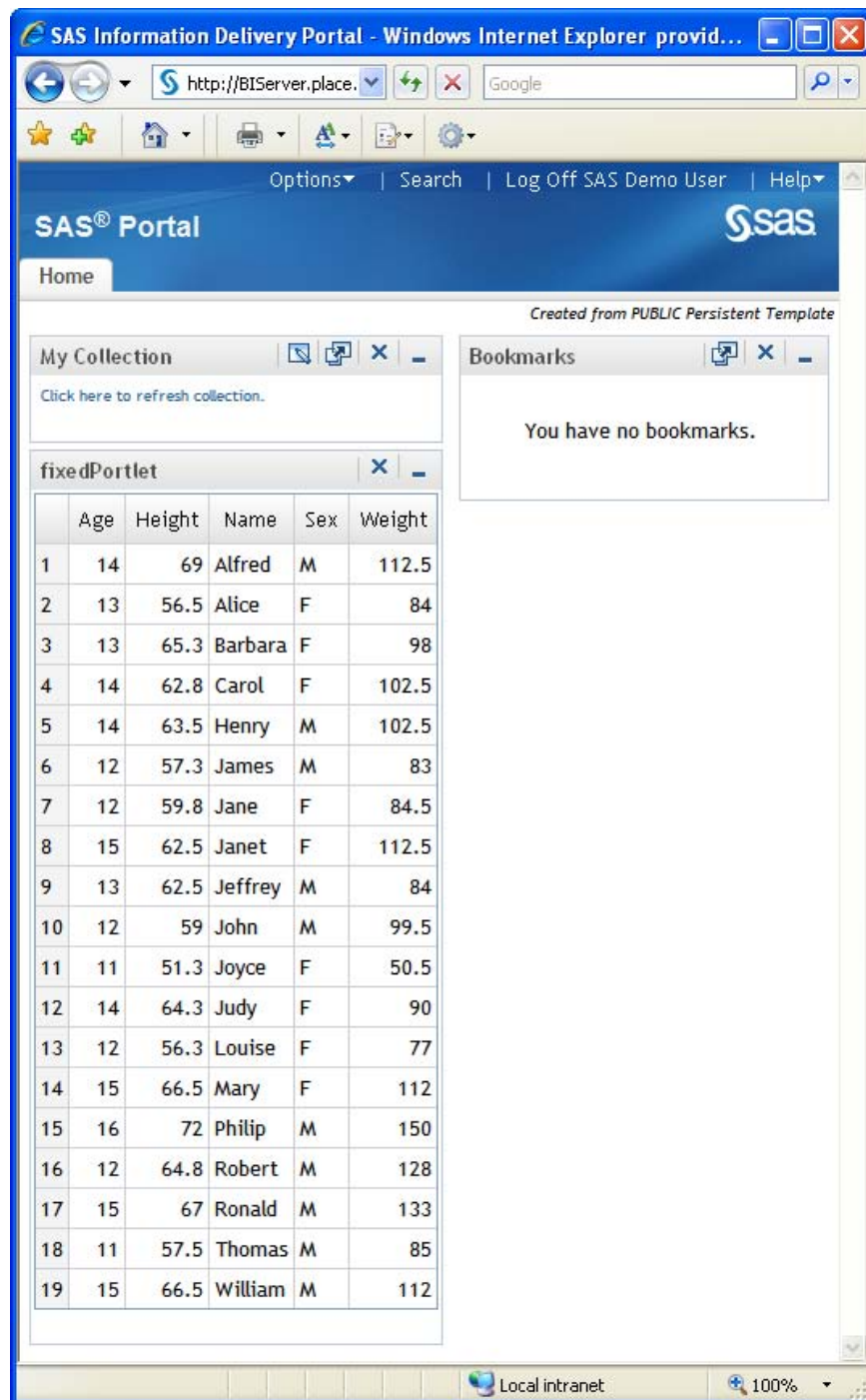
The par file is created, appears in the file list in Eclipse, and is copied to the `\Deployed` directory. The Par filename is created with the pattern `<projectName>.par`. In this case, the Par file is named `fixedPortlet.par`.

### View the Portlet in the SAS Information Delivery Portal

1. Navigate to your SAS Information Delivery Portal. For example: `BIServer.place.com:8080/SASPortal`
2. Select **Options** ⇒ **Edit Page Content**.



3. Click **Add Portlets**.
4. For the **Portlet type** select `fixedPortlet`.
5. For the **Name**, enter `fixedPortlet`.
6. Click **Add**, **Done**, and then **OK** to confirm your choices and return to the portal Home.
7. The portlet is displayed in the SAS Information Delivery Portal.



## The Portlet Editor

The portlet editor is a graphical interface for editing the `portlet.xml` file. You can open the portlet editor by double-clicking a `portlet.xml` file.

Using the portlet editor, you can specify several portlet options, and add parameters and actions. Actions are callbacks that the portal executes depending on the situation. For example, when a portlet is displayed the default action is executed. When you create an

action, the portlet editor creates a stub and adds it to the `portlet.xml` file. For information about portlets and actions, see *Developing Portlets for the SAS Information Delivery Portal* at <http://support.sas.com/documentation/online/doc/portal/index.html>.

**Portlets:**

List of portlets found in this portlet.xml file.

fixedPortlet (Local)

**Portlet Details:**

Parameters set for the portlet selected in the list.

**General:**

Name: fixedPortlet

Title: fixedPortlet

Editor Type: ☒ None ☐ Portal ☐ Portlet

☐ Pass Context ID

☐ Show Edit Properties

Initializer: \* fixedportlet.infomap.Initializer

**Deployment:**

Scope: ☒ User ☐ Group

Group:

☒ Auto Deploy

☒ User Can Create More

**Parameters:**

Name	Value
display-page	Viewer.jsp
map-name	SBIP://METASERVER/Public/CLASS(InformationMap)

Add... Edit... Remove

**Actions:**

Name	Type/URL	Default
display	fixedportlet.infomap.actions.Display	Yes

New... Link... Edit... Remove

Properties Source



## Chapter 6

# Template and Testing Details

---

<b>Files Added By the Metadata Creation Template</b> . . . . .	<b>45</b>
Location of Application Metadata . . . . .	45
The Launch Files . . . . .	46
Application.xml . . . . .	46
build.xml . . . . .	46
destination_omr.properties . . . . .	46
omr.properties . . . . .	46
<b>Copying the Application Metadata</b> . . . . .	<b>47</b>
<b>Files Added by the Stored Process Java Client Template</b> . . . . .	<b>47</b>
Introduction . . . . .	47
StoredProcessDriver.java . . . . .	47
StoredProcessConnection.java . . . . .	47
StoredProcessFacade.java . . . . .	48
StoredProcessDriver.properties . . . . .	48
<b>Files Added by the Stored Process Servlet Template</b> . . . . .	<b>48</b>
Introduction . . . . .	48
StoredProcessDriverServlet.java . . . . .	48
StoredProcessConnection.java . . . . .	49
StoredProcessFacade.java . . . . .	49
<b>Tomcat Configuration Details</b> . . . . .	<b>49</b>
Tomcat Support . . . . .	49
Configuring a Tomcat Server . . . . .	49
Initial Publish to Server . . . . .	50
Tomcat Shutdown Issue . . . . .	50
<b>Deployment and Authentication</b> . . . . .	<b>50</b>

---

## Files Added By the Metadata Creation Template

### *Location of Application Metadata*

When you add the SAS Web Infrastructure Platform Metadata Creation template to a project, and then you use that template to create metadata, files are added to the project. These metadata files are stored in the `\metadata` folder or one of its subdirectories.

## The Launch Files

When you add the Metadata Creation template to a project, three Ant configuration files (**.launch**) are added to the project. Their names indicate the task that they perform when executed:

- **ProjectName Copy Metadata.launch**

See [“Copying the Application Metadata” on page 47](#).

- **ProjectName Create Metadata.launch**

For instructions on how to do this, see [“Run the Launch File and Create the Application Metadata” on page 19](#) in the walk-through.

- **ProjectName Delete Metadata.launch**

## Application.xml

This file contains values that are used in the application metadata. For example, the name of the application is stored in the `<Name>` element, and the `<Context>`, `<Protocol>`, `<Host>`, and `<Port>` element values are combined to form the URL that the SAS Web Infrastructure Platform’s Logon Manager returns to the Web application after a successful login: `<Protocol>://<Host>:<Port><Context>`. The slash that follows the Port in the URL is part of the Context value.

You can edit the information in **Application.xml**, if necessary, but each time you run the metadata creation operation, the metadata for the application is replaced if it already existed.

The template sets the value for the Context element using the current value of the Context Root for the project. This value can also be found in project’s properties, on the Web Project Settings page. The Context value must match the name of the project.

The file is eventually transformed into the required deployment files and then those files are used to deploy the metadata.

## build.xml

This file is an Ant build file that supports the launch file operations.

## destination\_omr.properties

This file contains properties that specify the destination metadata server when copying the application metadata. The values in this file are placeholders that must be manually entered once a destination metadata server has been chosen. This file is not overwritten by repeated additions of the metadata template.

The **copy.can.replace.metadata** property controls whether existing metadata in the destination SAS Metadata Server is deleted before the copy. If false (the default), then the copy is not performed if metadata already exists in the destination metadata server for that application.

## omr.properties

This file contains properties that specify the destination metadata server when creating and deleting the application metadata, and the source metadata server when copying the

application metadata. The values in this file are determined by the SAS BI Server Profile that you selected when adding the template.

---

## Copying the Application Metadata

When moving to a production environment, copy the application metadata from the SAS Metadata Server where it was created to a destination metadata server by following these steps:

1. Enter the information for the destination metadata server in `\metadata\destination_omr.properties`.  
  
The `copy.can.replace.metadata` property controls whether existing metadata in the destination SAS Metadata Server is deleted before the copy. If false (the default), then the copy is not performed if metadata already exists in the destination metadata server for that application.
2. Right-click the *projectName Copy Metadata.launch* file, and select **Run As** ⇒ *projectName Copy Metadata.launch*.
3. Verify that BUILD SUCCESSFUL appears at the end of the output logged to the Console.

The copy operation extracts application metadata directly from the source metadata server and deploys it to the destination metadata server. No values in `Application.xml` or in the files derived from `Application.xml` when the application metadata was created are accessed. Consequently, changes made to the metadata using the SAS Management Console are included in the copy operation.

---

## Files Added by the Stored Process Java Client Template

### Introduction

When you add the “Java client for executing a SAS Stored Process” template to a SAS Java Project, Java files and one properties file are added to the source folder for the project.

### *StoredProcessDriver.java*

This class is an example of a Java console application client for executing a SAS Stored Process and writing the stored process results (and optionally the SAS Log) to a file. The default name of the class is `StoredProcessDriver`, but the actual name is derived from the class name for the client that was entered in the wizard.

### *StoredProcessConnection.java*

A `StoredProcessConnection` contains the information needed to use SAS Foundation Services, a SAS Metadata repository, and the SAS Stored Process Service for executing SAS Stored Processes. The class initializes and connects to SAS Foundation Services running in a Remote Services environment in preparation for executing SAS stored processes, and binds a `StoredProcessFacade` with the `StoredProcessDriver`.

SAS does not support customer implementations or extension of this class. The `StoredProcessConnection` generated for a SAS Java Project cannot be used in a SAS Web Application Project.

### ***StoredProcessFacade.java***

A `StoredProcessFacade` represents a SAS Stored Process and the results of its execution after the stored process has been executed. The `StoredProcessFacade` class wraps two interfaces from the SAS Stored Process Service:

`com.sas.services.storedprocess.StoredProcess2Interface` and  
`com.sas.services.storedprocess.Execution2Interface`.

This class can supply input parameters for the SAS Stored Process. Output parameter results or streaming results can be retrieved after the stored process is executed. The `StoredProcessFacade` provides other API methods for common operations, including the ability to access the underlying `StoredProcess2Interface` and `Execution2Interface` objects.

For more information, see the SAS Foundation Services Javadoc contained in the SAS BI API Documentation on the SAS AppDev Studio 3.4 Developer's Site (<http://support.sas.com/rnd/appdev/>), in particular the package summary for `com.sas.services.storedprocess`.

SAS does not support customer implementations or extension of this class. The `StoredProcessFacade` generated for a SAS Java Project cannot be used in a SAS Web Application Project.

### ***StoredProcessDriver.properties***

This file contains information about the SAS Metadata Server, and specifies log and output filenames for the results of executing the SAS Stored Process. These values can be changed so that the generated client can use a different SAS Metadata Server. The default name of this file is `StoredProcessDriver`, but the actual name is derived from the class name for the client that was entered in the wizard.

---

## **Files Added by the Stored Process Servlet Template**

### ***Introduction***

When you add the SAS Stored Process Servlet template to a SAS Web Application Project, Java files are added to the source folder for the project.

### ***StoredProcessDriverServlet.java***

The `StoredProcessDriverServlet` is an example of a servlet implementation for executing a SAS Stored Process, displaying the results in the browser, and optionally writing the contents of the SAS Log to the `ServletContext`. The default Example Base Name for a SAS Stored Process servlet is `StoredProcessDriver`. The actual name of the class is derived from the Example Base Name that was entered in the wizard.

**StoredProcessConnection.java**

A `StoredProcessConnection` contains the information needed to use SAS Foundation Services, a SAS Metadata repository, and the SAS Stored Process Service for executing SAS Stored Processes. The class binds a `StoredProcessFacade` with the `StoredProcessDriverServlet`.

SAS does not support customer implementations or extension of this class. The `StoredProcessConnection` generated for a SAS Web Application Project cannot be used in a SAS Java Project.

**StoredProcessFacade.java**

A `StoredProcessFacade` represents a SAS Stored Process and the results of its execution after the stored process has been executed. The `StoredProcessFacade` class wraps two interfaces from the SAS Stored Process Service:

`com.sas.services.storedprocess.StoredProcess2Interface` and  
`com.sas.services.storedprocess.Execution2Interface`.

This class can supply input parameters for the SAS Stored Process. Output parameter results or streaming results can be retrieved after the stored process is executed. The `StoredProcessFacade` provides other API methods for common operations, including the ability to access the underlying `StoredProcess2Interface` and `Execution2Interface` objects.

For more information, see the SAS Foundation Services Javadoc contained in the SAS BI API Documentation on the SAS AppDev Studio 3.4 Developer's Site (<http://support.sas.com/rnd/appdev/>), in particular the package summary for `com.sas.services.storedprocess`.

SAS does not support customer implementations or extension of this class. The `StoredProcessFacade` generated for a SAS Web Application Project cannot be used in a SAS Java Project.

---

## Tomcat Configuration Details

**Tomcat Support**

SAS AppDev Studio 3.4 continues to support Tomcat as a servlet container for testing SAS Web applications. But because Tomcat is not officially supported by SAS 9.2, Tomcat is recommended only for initial functional testing. Final acceptance testing should be done with the supported Web server that you plan to use with SAS 9.2. In addition, because the supported version of JBoss uses a version of Tomcat 6.0.x as its servlet container, a version of Tomcat 6 is recommended when using Tomcat for testing.

**Configuring a Tomcat Server**

If you have multiple SAS 9.2 BI installations that you want to test against, you can create separate Tomcat servers, each with the appropriate startup parameters to tie it to a corresponding BI installation. However, because each Tomcat server must have a unique name and because the name of the server must be hardcoded into the cheat sheet to make its initial configuration work correctly, when you configure additional servers you must manually open the `server.xml` file for that server to configure the UTF-8 support. Access

the “Configure a Tomcat Server for Testing” task through the “New Workspace Setup” cheat sheet. You can run the task independently of the larger cheat sheet. Instructions for when to manually edit the `server.xml` file are in the cheat sheet.

Enabling Java security is neither required nor recommended because of performance issues, and is not supported.

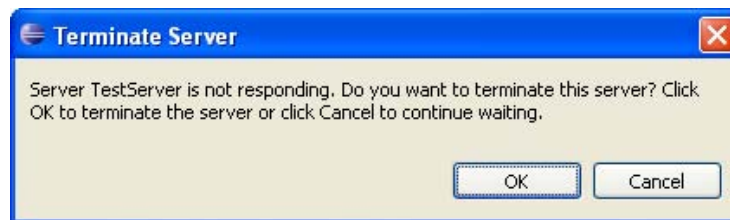
### Initial Publish to Server

The first time an application is published, all the static content and the jars are copied to the server. This content is over 100 megabytes, and the time it takes to copy it can exceed the default startup time-out of 45 seconds, resulting in a failure to start the server. Subsequent publishes only copy files that have changed.

Although you can adjust the startup time-out in the server properties, this will not guarantee that the time-out won't be exceeded as bandwidth can vary.

### Tomcat Shutdown Issue

Tomcat can fail to shutdown. The problem is that when you stop Tomcat, the Web application's classloader is also stopped before the necessary communications with Remote Services are completed. When Tomcat fails to shutdown within the stop time-out, a dialog box appears asking if you want to terminate the process. Terminating Tomcat from this dialog box is the only way stop the server in this situation.



The Restart command does not handle the time-out issue, and leaves the server in a permanent “Stopping” state without ever displaying the dialog box asking you if you want to terminate the server. When using the restart command and this happens, you must either restart Eclipse or manually kill the Tomcat process. Instead of using the Restart command from the Servers tab or when you confirm a restart when using **Run As** ⇒ **Run on Server**, you should separately Stop and then Start the server.

Although some servers might offer you the ability to omit restarting, when a class on the server changes, you should stop and then start the server so that the updated files are picked up by the server. Doing this also avoids potential memory leaks.

---

## Deployment and Authentication

The SAS Web Infrastructure Platform facet manages both the Web Infrastructure Platform Configuration and the support jars for a SAS Web application. Handling both enables the facet to keep the configuration features synchronized with the jars that support those features.

The SAS Web Infrastructure Platform configuration consists of static files found in `\WEB-INF\spring-config`, `context-param` declarations, and filter and servlet declarations and mappings. There is nothing in the configuration that binds the Web

application to a specific SAS BI installation. This means that the Web application can be deployed, unchanged (with one exception), to any SAS 9.2 BI installation, provided that the application metadata for the Web application and any resources the Web application depends on are present in that BI installation.

The one exception is the **parentContextKey** context parameter, which is found in the **web.xml** file. This parameter specifies whether an authentication token is in use. If the Remote Services for the SAS BI installation does not use an authentication token, then the value of **parentContextKey** must be **config.context**. If the Remote Services does use an authentication token, then the value must be **secure.config.context**.

Adding to a project a SAS Web application template that uses the SAS Web Infrastructure Platform alters the value of **parentContextKey** according to the presence of an authentication token in the BI Server Profile that you select.



## Chapter 7

# Exporting Projects

---

Exporting Java and SAS Java Projects as a Set of Jars .....	53
Exporting a SAS Web Application Project as a WAR File .....	54
Exporting a Project Using a Deployment Descriptor File .....	55

---

## Exporting Java and SAS Java Projects as a Set of Jars

The only projects that can be exported as a jar or set of jars are Java and SAS Java Projects that have a class with a `public static void main(String[])`.

To export a SAS Java Project, follow these steps:

1. Select **File** ⇒ **Export**.
2. Select **SAS Java Project** as the export destination, and then click **Next**.
3. Select the project to deploy.  
All of the projects available in the current workspace are listed. Invalid projects are indicated when selected.
4. Choose the resources to include in the output jar.  
Compiled code is not included in the tree. Items that are on the project source path are checked by default.
5. Select the **Include Source Code** check box if you want the project source code to be included in the exported jar.
6. Specify the **Deployment directory**.  
The jar containing the project code and other necessary files will be placed in this directory.
7. Specify the **Jar Name** and then click **Next**.
8. Select the **Use SAS Repository** check box if you want to create a jar that runs against a copy of the SAS Versioned Jar Repository that is already on the target machine.

If you do not deploy against a SAS Versioned Jar Repository, then all the required jars from the project and any dependencies from the SAS Versioned Jar Repository are copied into the deployment directory. You can then copy all the jars to a deployment location if necessary.

If you do deploy against a SAS Versioned Jar Repository, the project contents, a picklist called **SASRepositoryConfig**, and a script (in batch and shell versions) are copied to the deployment directory. The scripts use as their base name the name of the exported jar name. You must either edit the script and specify the location of the SAS Versioned Jar Repository on the target machine, or pass the location as a command-line argument. For example:

```
exportedJarName.sh repositoryLocation
```

The script executes the application using the SAS launcher to load the jar dependencies from the target machine's SAS Versioned Jar Repository.

9. Select the **Main class** that starts your application. The main class must reside in your project.
10. Click **Next**.
11. Select the **Save Deployment Descriptor in Project** check box if you want to capture the current export settings.

If you choose this option, specify the **Descriptor Location**, relative to the project. The filename must end in **.depdesc**. For more information, see [“Exporting a Project Using a Deployment Descriptor File” on page 55](#).

12. In the **Manifest Location** field, specify the path to the manifest file relative to the project.

The manifest file can be in any directory and with any filename. However, when added to the exported jar, the manifest information is placed in the file

**META-INF\MANIFEST.MF**.

If the specified manifest file already exists, the export overwrites any existing properties in the file that conflict with the properties needed to run the application properly. The overwriting occurs at the level of individual properties; the entire file is not overwritten.

13. Click **Next** and review the export settings, and then click **Finish**.

The project is exported to the specified deployment directory. Depending on the options that you chose, you might need to take additional configuration steps to run the project.

---

## Exporting a SAS Web Application Project as a WAR File

To deploy a SAS Web Application Project to a server outside of Eclipse, you must create a WAR file for the Web application contained in the project. This is accomplished using Eclipse's Export feature. Perform the following steps to create the WAR file:

1. Right-click the SAS Web Application project and select **Export** ⇒ **WAR file**.
2. Specify the **Destination** for the WAR file.
3. Select a target server run time, or clear the **Optimize for specific server runtime** check box.
4. Click **Finish**.

The resulting WAR file can be deployed to any application server or servlet container that provides full support for the Servlet 2.4/JSP 2.0 standard using JDK 1.5.0\_15 or higher for middle-tier development. However, only certain combinations of system hardware, JDK, and server have been tested by and are supported by SAS. Before deploying to a production

environment, consult the Third Party Software for SAS 9.2 Foundation document available at <http://support.sas.com/resources/thirdpartysupport/v92/index.html>. Specifically, examine the JDK and Application Servers sections for details about the supported combinations and the server security fixes that are expected to be installed.

Additional information about Web application deployment is available on the SAS AppDev Studio Developers Site at <http://support.sas.com/rnd/appdev/>.

---

## Exporting a Project Using a Deployment Descriptor File

Deployment descriptor files contain the export settings for a project. You can save a descriptor file when you export a project. On subsequent exports you can use the descriptor file to bypass the Export dialog box. You can have multiple descriptor files per project. Descriptor files must end with **.depdesc**.

To use a descriptor file to bypass the Export dialog box, expand the project in the Package Explorer or Navigator, right-click the descriptor file, and select **Export SAS Project**. The project is exported using the settings in the descriptor file.

To use a descriptor file to populate the Export dialog box with the settings in the file, right-click the file and select **Open SAS Export**.

To inspect or change the settings in a descriptor file, you can open the file as XML by double-clicking it, and editing values as needed. Unless otherwise specified, the first descriptor file listed is used to populate the Export dialog box.



## Chapter 8

# Managing the Jars in a Project

---

<b>The SAS Repository</b> .....	<b>57</b>
<b>Opening the SAS Repository Properties Editor</b> .....	<b>58</b>
<b>Identifying Dependent Jars</b> .....	<b>59</b>
<b>Removing Jars from the Classpath</b> .....	<b>59</b>
<b>Changing the Order of Jars in the Classpath</b> .....	<b>60</b>
<b>Adding New Jars to the Classpath</b> .....	<b>61</b>
<b>Adding Dependent Jars</b> .....	<b>61</b>
<b>Specifying the Current Versions of Jars</b> .....	<b>62</b>
<b>Specifying Other Jar Versions</b> .....	<b>62</b>
<b>Removing Version Restrictions</b> .....	<b>63</b>
<b>Reporting the Classpath Jars</b> .....	<b>63</b>
<b>Reporting Jar Relationships</b> .....	<b>64</b>
<b>Finding a Class in a Jar</b> .....	<b>65</b>
<b>Changing Default Classes for SAS Java Projects</b> .....	<b>66</b>

---

## The SAS Repository

There are two pieces to managing the jars that a project requires: the SAS Versioned Jar Repository, and the project's SAS Repository.

The SAS Versioned Jar Repository is a common storage location for all the jars that are supplied by the installed SAS products. This includes some third-party jars. This repository is incrementally updated as SAS products are installed.

A project's SAS Repository is a configurable classpath container that is added to the Java build path of all SAS projects to provide access to the jars in the SAS Versioned Jar Repository. The SAS Repository determines which SAS jars need to be included in a project's build path. It can also determine any jar dependencies and automatically include those dependencies in the project's Java build path. This ability means that you need to specify only the primary jars that a SAS project requires, the dependencies are handled for you.

When necessary, you can bypass the SAS Repository automation and specify exactly which jars, and which versions of each jar, to include. This flexibility enables you to manage jar

dependencies on a per-project basis without disrupting past development, and contributes to a smaller distribution footprint.

If you want to add to the project a jar that is not available in the SAS Versioned Jar Repository, open the project properties, select **Java Build Path**, and use one of the mechanisms that does not involve editing the SAS Repository on the **Libraries** tab. You are responsible for adding any jar dependencies.

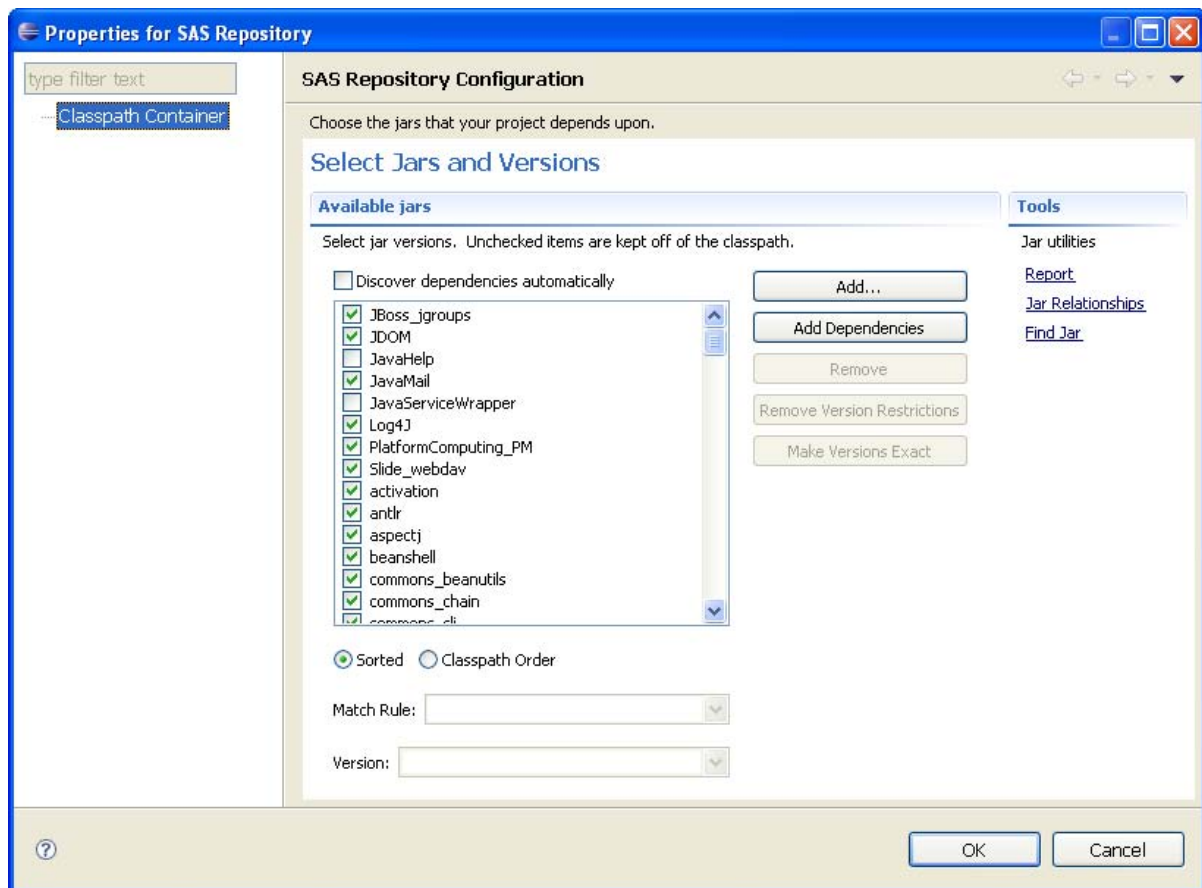
## Opening the SAS Repository Properties Editor

Access to the SAS Repository properties depends on the project type and the current perspective. When in the Java perspective, all project types (SAS Java projects and SAS Web Application projects) display the SAS Repository at the root of the project. To access the Properties for SAS Repository dialog box, follow these steps:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.

The Properties for SAS Repository dialog box appears.

When in the Java EE perspective, the SAS Repository is located at the top level of SAS Java Projects, and in `\sourceFolder\Libraries\` for SAS Web Applications projects.



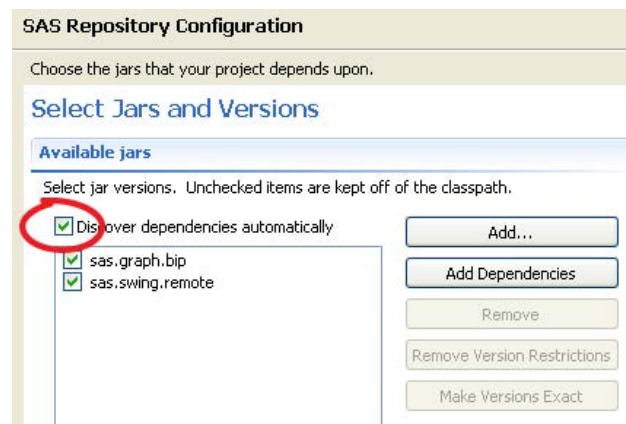
You can also access the SAS Repository as a library:

1. Right-click the project and select **Properties**.
2. Select **Java Build Path**, the **Libraries** tab, then **SAS Repository**, and then click **Edit**.

---

## Identifying Dependent Jars

To automatically add the dependencies of all checked jars to the classpath, select the **Discover dependencies automatically** check box. With this option selected, the classpath is automatically updated if there are changes in the SAS Versioned Jar Repository that affect the jars required by your project.



If you click the **Add Dependencies** button, all the jar dependencies for the jars that are checked in the **Available jars** list are added to the Available jars list. This enables you to specify a version for each of the dependent jars, if needed.

If the **Discover dependencies automatically** check box is not selected, only the checked jars are added to the classpath. To exclude a jar from the classpath, uncheck it. A jar that is not checked is not passed to the compiler or run-time engine.

You should use the **Discover dependencies automatically** option only if you are comfortable with jars being automatically added to the classpath, knowing that updates to the SAS Versioned Jar Repository might change which jars are required for your project.

For a SAS Web Application project, the **Discover dependencies automatically** option is unchecked by default. Using this option is not recommended for SAS Web Applications because the jars in the project must be compatible with the static content provided by the SAS Web Infrastructure Platform. The necessary jars are specified internally.

---

## Removing Jars from the Classpath

To remove one or more jars from your project's classpath, follow these steps:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.

The Properties for SAS Repository dialog box appears.

3. Select the jar or jars to remove and then click the **Remove** button.

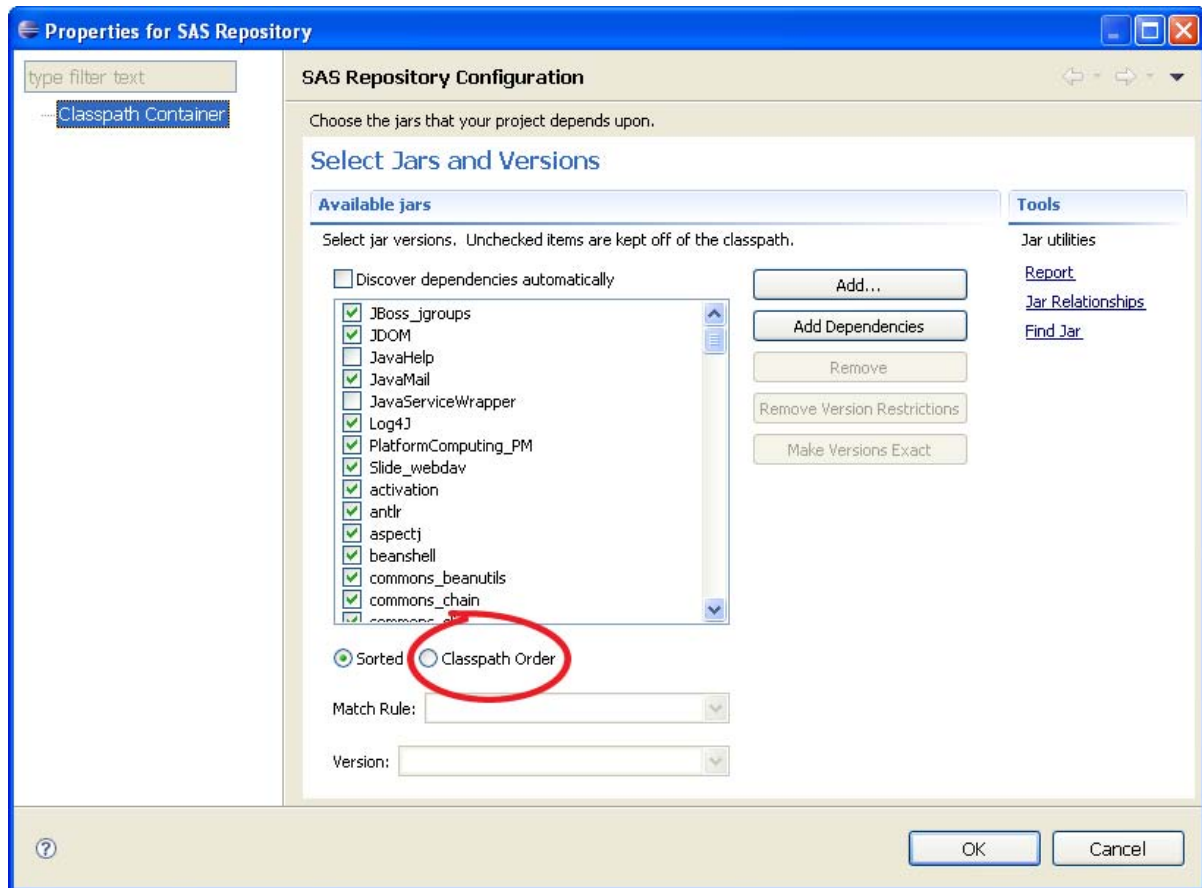
Note that using the Organize SAS Imports action adds a removed jar to the classpath (see “Using the Organize SAS Imports Action” on page 69 ). In contrast, when you uncheck a jar in the **Available jars** list (see “Identifying Dependent Jars” on page 59 ) the Organize SAS Imports command does not add the unchecked jar back to the classpath.

## Changing the Order of Jars in the Classpath

The order of the jars in the **Available jars** list determines the order that the jars appear in the classpath. The higher a jar is listed in the interface, the closer to the front of the classpath it appears.

To change the order of the jars, follow these steps:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.  
The **Properties for SAS Repository** dialog box appears.
3. Select **Classpath Order**.



4. Select a jar in the **Available jars** list and click the up or down arrow.

---

## Adding New Jars to the Classpath

To add new jars to the classpath, follow these steps:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.

The Properties for SAS Repository dialog box appears.

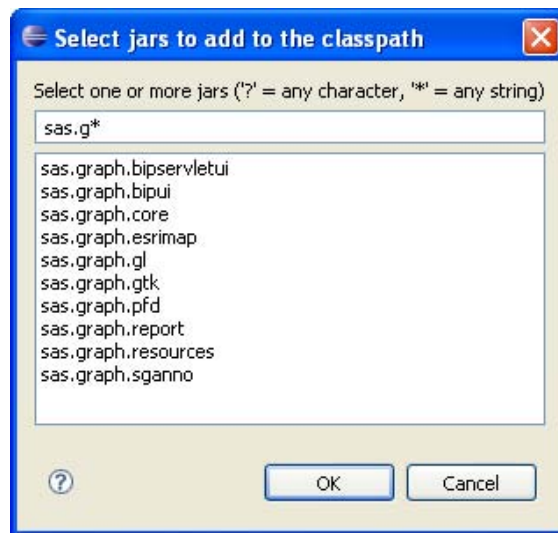
3. Click **Add**.

A dialog box appears that lists all the jars that are available in the SAS Versioned Jar Repository.

4. Select one or more jars and click **OK**.

To narrow the list of available jars, enter a substring in the text field at the top of the dialog box. Use the regular expression characters provided.

A second dialog box might appear listing jars that are required by the jars you explicitly selected. Click **Yes** to add those jars, or **No** to add only the jars that you explicitly selected.



---

## Adding Dependent Jars

The **Add Dependencies** button determines whether there are missing jars that the project's current jars require. This is useful when the **Discover dependencies automatically** is not selected, and you need to find missing jars, or **Discover dependencies automatically** is selected but you need to find a dependency so that you can override the version. Follow these steps to identify dependent jars:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.

The Properties for SAS Repository dialog box appears.

3. Click **Add Dependencies**.

If a missing jar is detected, the latest version of the jar in the repository is appended to the list of selected jars.

---

## Specifying the Current Versions of Jars

Specifying the current version of a jar (as opposed to the most recent) is useful for removing any ambiguity about which version of a jar is specified on the classpath. To specify that the current version of a jar should always be used, and you do not want to use a new version when it becomes available, follow these steps:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select Properties.

The Properties for SAS Repository dialog box appears.

3. Select the jar or jars and click **Make Versions Exact**.

For each selected jar, the **Match Rule** is changed to **Exact**, and the **Version** is changed to the version of the jar currently in use.

---

## Specifying Other Jar Versions

You can control which jars get used with varying degrees of specificity. Eclipse jar versions follow the pattern **Major\_version.Minor\_version.Micro\_version.tag**, where **tag** represents characters used to differentiate versions. If more than one version of a jar that matches your specifications is found in the repository, the latest version is used. You can have only one version of a jar in your classpath.

To specify a version of a jar that is not the latest version, follow these steps:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.

The Properties for SAS Repository dialog box appears.

3. Select the jar or jars to which you want to apply a versioning rule.
4. Select a **Match Rule**, as follows:

- **Latest** - The highest version of the jar.
- **At Least** - The highest version of the jar that is at least the version selected in the Version list.
- **Exact** - The jar specified in the Version list. No higher or lower version can be substituted. Only for an Exact match is the tag of the version tested.
- **Up To** - The highest version of the jar up to and including the version in the Version list.
- **Latest Within Major Version** - The jar with the highest version but with the same major version specified in the Version list.

- **Latest Within Minor Version** - The jar with the highest version but with the same major and minor versions specified in the Version list.
  - **Latest Within Micro Version** - The jar with the highest version but with the same major and minor and micro versions specified in the Version list.
5. Select the **Version**.

If the currently selected **Match Rule** requires version information, select or enter the version number in the **Version** combo box, which is populated with all of the version numbers that are currently available in the SAS Versioned Jar Repository. Note that the jars themselves might have their own version requirements for dependent jars.

---

## Removing Version Restrictions

To remove version restrictions on a jar and get the latest version, follow these steps:

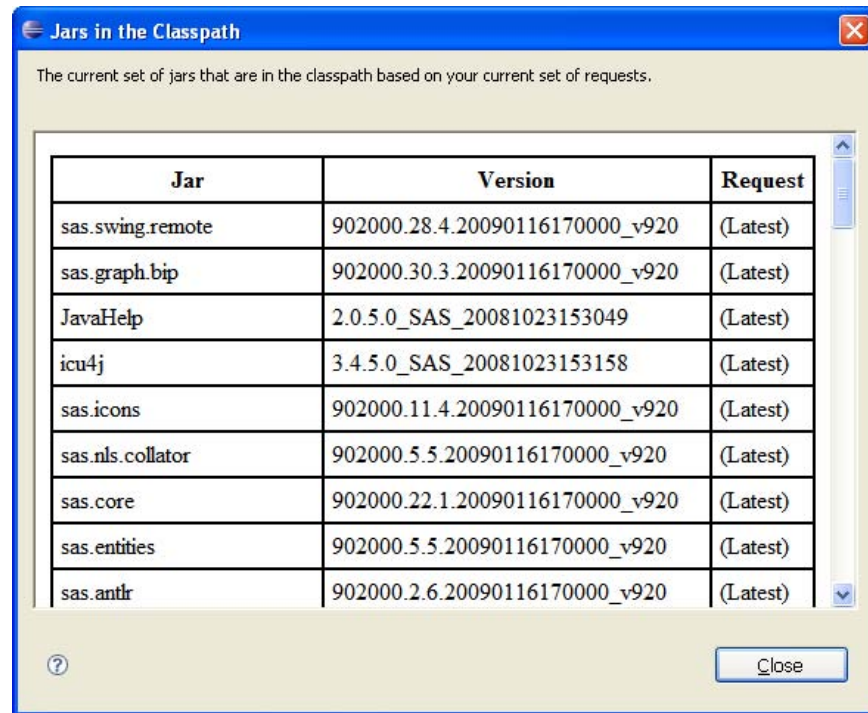
1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.  
The Properties for SAS Repository dialog box appears.
3. Select the jar and click **Remove Version Restrictions**.  
The **Match Rule** is changed to **Latest**, and the **Version** is cleared.

---

## Reporting the Classpath Jars

To generate a report of all the jars that will be used on the classpath, including the reason that a specific version of a jar version was included, follow these steps:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.  
The Properties for SAS Repository dialog box appears.
3. Click **Report**.



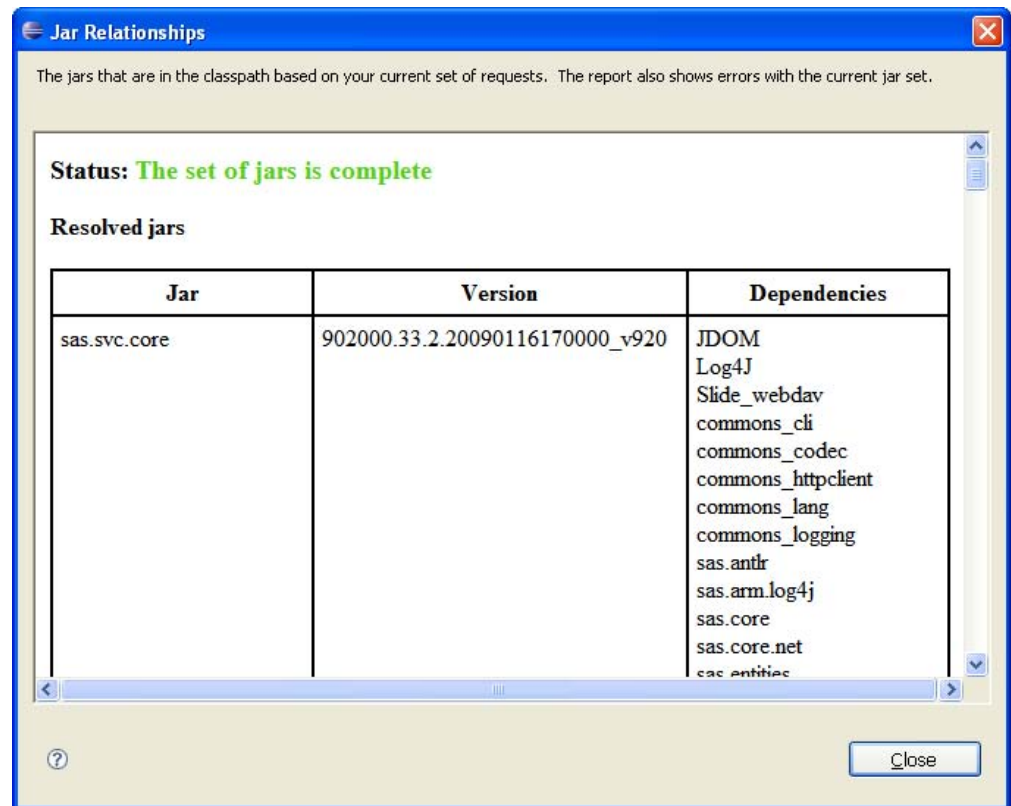
## Reporting Jar Relationships

To generate a report showing the relationship between the jars in the project's SAS Repository, and which jars are missing or have constraints that can't be satisfied, follow these steps:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.  
The Properties for SAS Repository dialog box appears.
3. Click **Jar Relationships**.

The resulting report can include the following sections:

- **Resolved jars** – lists the name and version of each jar that was included and also the names of dependent jars that the included jar requires.
- **Missing jars** – lists jars that are required by the jars that are currently checked in the Available Jars list, but are not currently included. The jar IDs are listed, along with the name of the jar that directly requested them.
- **Unresolved jars** – Displays the version request string and the likely error.



## Finding a Class in a Jar

To find a class in your project's jars, and then add that jar to the SAS Repository and the classpath, follow these steps:

1. Expand the project in the Package Explorer.
2. Right-click **SAS Repository** and select **Properties**.

The Properties for SAS Repository dialog box appears.

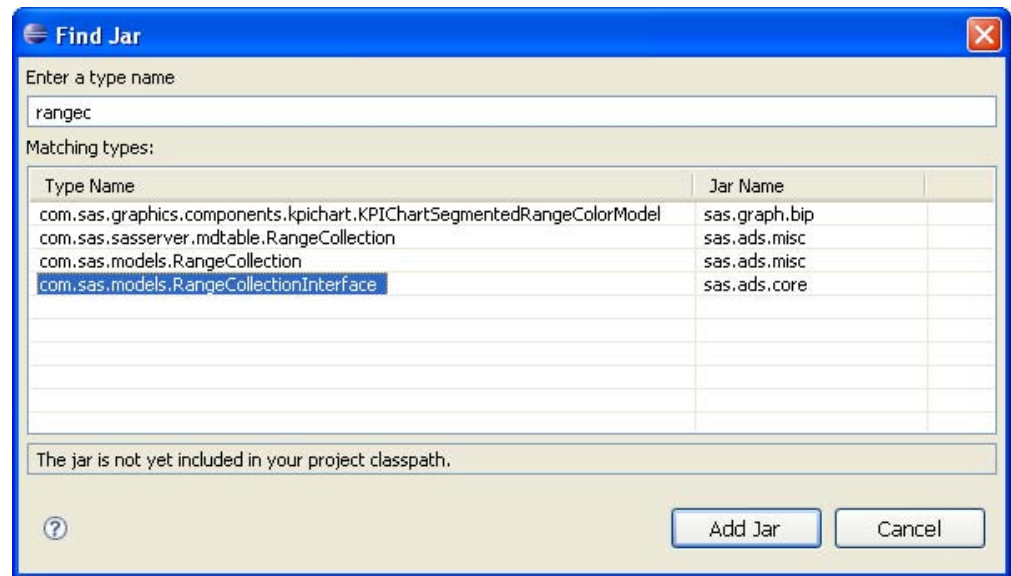
3. Click **Find Jar**.
4. Enter the local name of a class (not the package).

The search is case insensitive and returns all classes in the project's SAS Repository that contain the search string.

5. Select the class that you want from the list.

If the **Add Jar** button is enabled, then the jar is not in the project's SAS Repository. Click **Add Jar** to add it.

If the **Add Jar** button is dimmed, then the jar is already in the project's SAS Repository.



## Changing Default Classes for SAS Java Projects

By default, the classpath for a new SAS Java Project includes **sas.swing.remote**, **sas.graph.bip**, and all their dependent jars. To change the default classes that are added to a SAS Java project, follow these steps:

1. Select **Window** ⇨ **Preferences**.
2. Expand **SAS AppDev Studio**, and select **SAS Project dependencies**.
3. Change the included jars just as you would for a project.
4. Click **OK**.

New projects will use the new default jars. Existing projects are not affected.

## Chapter 9

# Using the SAS Editor Extensions

---

<b>Introduction to SAS Editor Extensions</b> .....	<b>67</b>
<b>Accessing SAS Component API Documentation</b> .....	<b>68</b>
Introduction .....	68
Eclipse Help System .....	68
Context-Sensitive Javadoc in an Eclipse Window (F1) .....	68
Context-Sensitive Javadoc in an External Browser (Shift+F2) .....	68
Configuring External Javadoc for use within Eclipse .....	68
<b>Adding Missing Import Statements</b> .....	<b>69</b>
Introduction .....	69
Using a SAS Import from Repository Quick Fix .....	69
Using the Organize SAS Imports Action .....	69
<b>Attaching a SAS Model to a Viewer</b> .....	<b>70</b>
Overview of Model/Viewer Connections .....	70
Using the SAS Model/Viewer Connection Quick Assist .....	71
<b>SAS Snippets</b> .....	<b>72</b>
Snippets Introduction .....	72
Displaying the SAS Snippets .....	72
Inserting a SAS Snippet .....	73

---

## Introduction to SAS Editor Extensions

The SAS AppDev Studio 3.4 Eclipse plug-ins add the following functionality to the default Eclipse set up:

- access to the SAS API documentation from code
- identification and addition of missing import statements
- assistance with model/view attachments
- SAS snippets

---

## Accessing SAS Component API Documentation

### Introduction

You can access the SAS Component API Javadoc from within Eclipse via the following:

- the Eclipse Help system
- context-sensitive Help in an Eclipse window (F1)
- context-sensitive Help in an external browser (Shift+F2)

### Eclipse Help System

To browse the SAS Component API documentation, follow these steps:

1. Select **Help** ⇒ **Help Contents**.
2. Expand **SAS Components Guide**.
3. Expand **Reference**.

### Context-Sensitive Javadoc in an Eclipse Window (F1)

To view the Javadoc for a class in an Eclipse window, place the text insertion point on a class name in the source code and press F1. A Help view appears in the workbench. The first link in the Help view is a link to the Javadoc, if Javadoc is available for the class.

The link in the Help view is context sensitive, and will change if you reposition the insertion point to a different class.

### Context-Sensitive Javadoc in an External Browser (Shift+F2)

If the class is on the build path of the project, you can view the Javadoc for the class in an external browser by placing the insertion point on the class name in the source code and then pressing Shift+F2.

### Configuring External Javadoc for use within Eclipse

To specify more current documentation for a jar, or to add a reference to Javadoc for a third-party jar, follow these steps:

1. Open a project that uses the jar.
2. In the Package Explorer, expand **SAS Repository**.
3. Right-click on the jar with which you want to associate Javadoc and select **Properties**.
4. Select **Javadoc Location** and then specify the location.

Javadoc for SAS Components is not displayed when you hover over a class name or method call in the source code because the Javadoc for tooltips is generated from source code, and the SAS source code is not distributed.

## Adding Missing Import Statements

### Introduction

When you declare or use a class that is in the classpath but is not yet imported into your Java program, Eclipse provides a Quick Fix that can add an import statement for that class to the top of the source file. Eclipse also provides an Organize Imports action that can add the necessary imports for a given source file. However, neither the Eclipse Quick Fix nor the Organize Imports action are aware of the SAS Versioned Jar Repository.

If you are using content from the SAS Versioned Jar Repository, use the SAS Import from Repository Quick Fix and the Organize SAS Imports action because both are aware of the SAS Versioned Jar Repository.

### Using a SAS Import from Repository Quick Fix

A **SAS Import from Repository Quick Fix** is displayed when there is a compilation error because a class used in the code cannot be resolved in the current source file.

If you accept the Quick Fix, the SAS Import from Repository Quick Fix adds the following items:

- the jar containing the class to the project classpath
- an import statement for the class to the current file.

If the name of the class (excluding the package) is included on the current project classpath, Eclipse provides a Quick Fix for each of the candidate class names. If there are additional candidate class names in the latest versions of one or more jars in the SAS Versioned Jar Repository, Quick Fixes are also provided for those class names.



To use one of the SAS Import from Repository Quick Fixes (marked with a SAS icon), double-click the Quick Fix or select the Quick Fix and press ENTER. If you use a SAS Import from Repository Quick Fix and the Build Automatically option is enabled, the entire project is rebuilt because the classpath has changed.

### Using the Organize SAS Imports Action

In cases where there are multiple unknown types used in a source file, the Organize SAS Imports action can guide you through the process of resolving all of the unknown classes (both classes from the SAS Versioned Jar Repository and other classes) in the current source file. If you add any new jars to the project dependencies, the project must be rebuilt.

A Java file that cannot be found in a class exists in one of two states:

- the class is currently on the project classpath, but it has not been imported into the Java file. In this case, an import statement is added to the file and a build of that particular file is required.
- the class is not currently on the project classpath. In this case, the classpath must be modified to include the required jar(s), and an import statement must be added to the file. A full build of the project is required because of the change to the classpath.

To start the Organize SAS Imports action, right-click in a Java file in the Java editor and select **Source** ⇒ **Organize SAS Imports**.

---

## Attaching a SAS Model to a Viewer

### Overview of Model/Viewer Connections

#### Introduction

SAS AppDev Studio 3.4 does not support attaching models to viewers using drag-and-drop, but there is an editor Quick Assist that facilitates model/viewer connections.

Note that the Quick Assist cannot determine whether you have already attached a model to a viewer. You can run the Quick Assist multiple times on the same viewer instantiation, each time attaching it to a different model. To determine which attachment is in effect, you must examine the code and identify the last attachment called.

#### Enabling the Quick Assist Icons

Eclipse uses a lightbulb icon in the editor gutter to notify you when a Quick Assist is available. By default, the Quick Assist icons are not displayed. To enable the Quick Assist icons, follow these steps:

1. Select **Window** ⇒ **Preferences**.
2. Expand **Java**.
3. Select the **Editor** item.
4. Select the **Light bulb for quick assists** check box.

#### Supported Viewers for Model/Viewer Connection Quick Assist

The Connection Quick Assist supports viewers that have a `setModel` method that takes as an argument one of the following types:

- `javax.swing.ComboBoxModel`
- `javax.swing.ListModel`
- `com.sas.table.StaticRowTemplateTableInterface`
- `com.sas.table.StaticTableInterface`
- `javax.swing.table.TableModel`
- `javax.swing.tree.TreeModel`

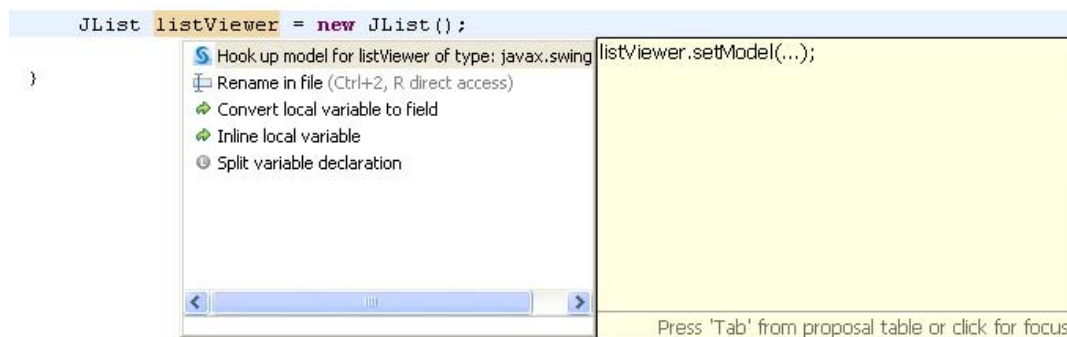
## Using the SAS Model/Viewer Connection Quick Assist

To use the SAS Model/Viewer Connection Quick Assist, follow these steps:

1. Place the cursor on a statement that contains a variable instantiation of a viewer type (for example, `javax.swing.JList`). The Quick Assist will not function if you place the cursor on a static method.
2. Press CTRL+I.

From the list of Quick Assist choices, select **Hook up model**.

You can also access the connection Quick Assist by right-clicking and selecting **Source** ⇒ **Add SAS Attachments**.



3. Select how you want to create and link a model to the viewer:

- **Use existing field**
- **Create new field**

If you selected **Use existing field**:

- a. Select the field to use.

Only the models declared as fields that are within the same scope as the cursor location and match the available model type(s) are listed. If there are no available fields, the **Use existing field** option is disabled.

- b. Click **Finish** to generate the code that creates the model and attaches it to the viewer.

If you selected **Create new field**:

- c. Select the model to attach to the viewer from the **Choose a class to declare** list.

When you expand an available class, the child nodes represent the classes in the current classpath that extend or implement the expanded class.

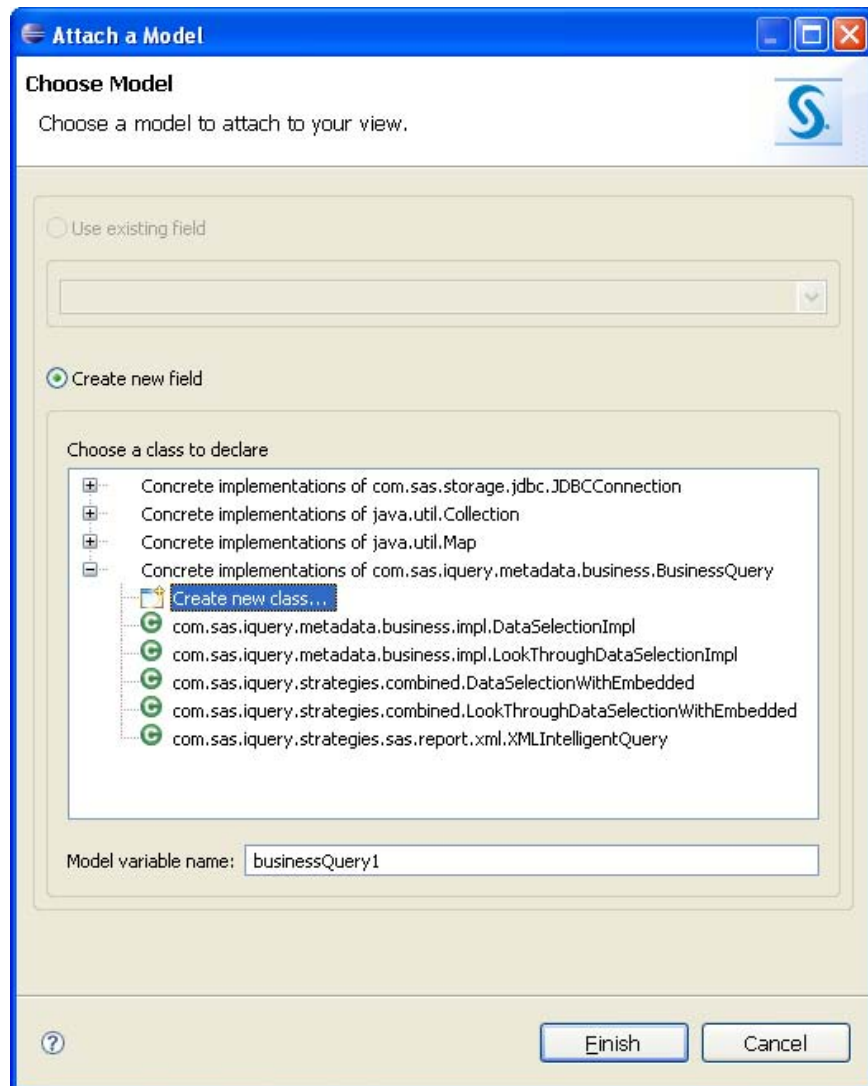
- d. Enter the **Model variable name**.

This variable name is used in the declaration in the source file. The variable is declared in the top-most enclosing class.

A default variable name is supplied, based on the currently selected class. If you modify this name, it will not change when you select a different class.

- e. Click **Finish** to add an initialization method to the source file with statements that instantiate the appropriate type for this variable (multiple fields are written with the same initialization method).

A secondary dialog box appears, giving you the opportunity to name and adjust other creation options for the new class before the initialization method is created.



## SAS Snippets

### *Snippets Introduction*

Snippets are pieces of code that perform some action for you. Typically they enable you to configure a piece of code and then insert it, saving you from a lot of typing. Snippets can also perform actions for you, like searching a SAS Metadata server. Snippets are a robust form of the Eclipse Editor snippets.

### *Displaying the SAS Snippets*

To display the SAS snippets, follow these steps:

1. Select **Window** ⇒ **Show View** ⇒ **Other**.

2. Expand **General**.
3. Select **Snippets** and then click **OK**.
4. Inside the **Snippets** tab, click on **SAS Snippets**.

### Inserting a SAS Snippet

To insert a snippet, follow these steps:

1. Place your cursor inside a non-static method body where you want to insert the snippet.
2. Double click on the snippet that you want to insert (or right-click and select **Insert**).
3. Configure the variables used in the snippet.

Variables in the snippet are either declared at the top of the snippet, or are replaced by variables that already exist in the code.

**Configure Snippet**

Specify how the variables in the snippet should be initialized

Choose an initializer for each snippet variable:

Purpose	Type	Initialization Code
Database URL	java.lang.String	ENTER_DATABASE_URL_HERE
Driver Name	java.lang.String	ENTER_DRIVER_NAME_HERE
User ID	java.lang.String	ENTER_USER_ID_HERE
Password	java.lang.String	ENTER_PASSWORD_HERE

Variable Description:

☐ Surround generated code with try/finally

Resulting source code:

```
String databaseURL=ENTER_DATABASE_URL_HERE; //The URL of the database. (ex. "jdbc:sasiom://hostname:859
String driverName=ENTER_DRIVER_NAME_HERE; //The class name of the driver (ex. "com.sas.rio.MVADriver")
String userID=ENTER_USER_ID_HERE; //The user ID to use to make the connection (ex. "<domain>{<userID>}")
String password=ENTER_PASSWORD_HERE; //The password to use to make the connection

JDBCConnection jdbcConnection_1 = new JDBCConnection();
Properties properties_1 = new Properties();
//set any additional properties for this connection here. See the documentation for the driver you are using
//for the names of relevant properties.
// For example, you can assign the librefs property
// on the com.sas.rio.MVADriver in the following manner
// properties_1.put( "librefs", "appdev 'c:/data/tables'");
properties_1.put(JDBCConstants.USER, userID);
properties_1.put(JDBCConstants.PASSWORD, password);
jdbcConnection_1.setConnectionInfo(properties_1);
jdbcConnection_1.setDatabaseURL( databaseURL);
jdbcConnection_1.setDriverName( driverName );
```

The **Initialization Code** field for each variable defines how the variable is assigned. The **Initialization Code** can be set one of three ways:

- An illegal value that will not compile so that you are forced to fix it. For example, **ENTER\_HOST\_NAME\_HERE**.

- Arbitrary text that you enter.

You can enter any text, method call, or other complex expression for the **Initialization Code**. If entering a string, include the appropriate quotation marks and escape characters.

- A variable from your code.

The list of values is populated with all of the variables in scope that are compatible with this variable type. If you select one of these variables, the declaration at the top of the snippet is removed and the variable that you have selected is used throughout the snippet.

4. Choose whether to **Surround generated code with try/finally**. This option encloses the generated code in a `try/finally` block. This is useful if you want to handle the declared exceptions in the current method.
5. Click **Insert**.

The snippet code is added at the cursor location and the necessary import statements are added to the top of your source file. Inserted type references are fully qualified if they would conflict with an existing import.

## Appendix 1

# Creating a SAS Web Application that Does Not Use the Web Infrastructure Platform

### Introduction

In AppDev Studio 3.4, a standard SAS Web Application project contains both the “SAS Java Components” facet and the “SAS Web Infrastructure Platform” facet. The SAS Web Application Project wizard always adds both facets when creating a new SAS Web application project. Once added, neither SAS facet can be removed from the project. Consequently, the SAS Web Application Project wizard cannot be used to create a SAS Web application project that does not include the “SAS Web Infrastructure Platform” facet.

There is, however, a process to create such a project that does not involve using the wizard. Before you create the project you must know whether you are deploying using Remote Services, Local Services, or both. Remote Services is the preferred method.

If you are using Local Services, you must create a Local Services Deployment Descriptor. Because none of the templates provided with AppDev Studio 3.4 use Local Services, if you want to use one of the provided templates with Local Services, you must modify the template.

No matter how you deploy, because the project does not use the Web Infrastructure Platform, it cannot integrate with the Logon Manager application.

### Create a Dynamic Web Project

Create a new SAS Web Application project that contains only the SAS Java Components facet by following these steps:

1. From Eclipse, select **File** ⇒ **New** ⇒ **Project**.
2. Expand the **Web** folder.
3. Select **Dynamic Web Project** and click **Next**.
4. Specify a project name.  
This name is used as the context name. The name must not contain spaces.
5. Select the **Target Runtime** to match the server that you plan to use.  
The default target run time is ADS Apache Tomcat v6.0.
6. For the Dynamic Web Module version, select **2.4**.
7. For the Configuration, select **SAS 9.2 Java Components Configuration**, and then click **Finish**.

### Create a Local Services Deployment Descriptor

If you are deploying using Local Services, create a Local Services Deployment Descriptor. AppDev Studio 3.4 does not provide an ADS Local Services deployment descriptor as prior releases did. Instead, you must duplicate the descriptor used by a SAS BI Web application.

1. Open SAS Management Console and logon as a SAS metadata administrator to connect to the SAS Metadata Server for the SAS BI installation that you are developing against.
2. Select the **Plug-ins** tab.
3. Under the Environment Management folder, expand **Foundation Services Manager**.
4. Right-click **SASWIPServices9.2 Local Services**, and then select **Duplicate Service Deployment**.
5. Enter a name for the service deployment that is appropriate for your Web application, for example “SASAppDevStudio3.4 Local Services”.
6. Expand the tree for the new service deployment, and then expand the **Core** folder.
7. Right-click **Logging Service**, and then select **Properties**.
8. Select the **Service Configuration** tab.
9. Click **Configuration**.
10. Select the **Outputs** tab.
11. From the Outputs list, select **SAS\_LS\_FILE**.
12. Click **Delete**.

Alternately, edit the log filename so that it is specific to your Web application. Ensure that the path to the log file exists. Initially this path is on the AppDev Studio system that is testing the migrated Web application project, but the path must also exist on the remote system when the application is deployed.

13. Click **OK** to confirm all actions and exit the SAS Management Console.

### Add a Context Listener

The last step in creating a SAS Web Application Project that does not use the Web Infrastructure Platform is to add a context listener that deploys the new Local Services or Remote Services.

However, if you are using one of the SAS Java Web Application templates provided with AppDev Studio 3.4, a Remote Services context listener is automatically added to the project on a successful build. In such a case, you need to add a context listener only if you are changing that template so that it uses Local Services.

To add the context listener, follow these steps:

1. From Eclipse, select **File** ⇒ **New** ⇒ **Other**.
2. Expand the **SAS AppDev Studio** folder.
3. Select **Add Template Content to Project**, and then click **Next**.
4. For the **Project**, select the host project that was created earlier.
5. Expand the **SAS Java Web Application** and then **SAS Foundation Services Support** folders.

6. If the Web application deploys only local SAS Foundation Services, select **Context Listener For Local Services**. If you are deploying only SAS Remote Services and are not adding a SAS Web Application Example template, select **Context Listener for Remote Services**. If the Web application also deploys SAS Remote Services, select **Context Listener for Remote and Local Services**.
7. Click **Next**.
8. Click **Next** to accept the default Template Configuration Parameters.
9. If using Local Services, select the BI Server Profile for the BI installation whose SAS Metadata Server holds the new local services deployment descriptor (created in the previous section).
  - a. Click **Advanced**.
  - b. In the Local Services Deployment section, select **Other**.
  - c. Click **Browse** and select the service deployment descriptor that you created earlier.
  - d. Click **OK**.
10. If the SAS Metadata Server for the BI installation is running, you can click **Test Configuration** to verify that the service deployment or deployments can be read from the metadata server.
11. Click **Finish**.



# Index

---

## A

- Accessibility features [5](#)
  - exceptions [5](#)
- API documentation [68](#)
- AppDev Studio
  - new features [7](#)
  - overview [7](#)
- application metadata
  - copying [47](#)
  - files [45](#)
  - launch files [46](#)
- Application.xml [46](#)
- authentication
  - and deployment [50](#)

## B

- BI Server profiles [10](#)
- build.xml [46](#)

## C

- classes
  - changing default for SAS Java Projects [66](#)
- classpath
  - adding dependent jars [61](#)
  - adding new jars [61](#)
  - changing jar order [60](#)
  - listing jars [63](#)
  - removing jars [59](#)
- Connection profiles [11](#)

## D

- dependent jars
  - identifying [59](#)
- deployment
  - and authentication [50](#)
- deployment descriptor [55](#)
- destination\_omr.properties [46](#)

- documentation
  - accessing [68](#)

## E

- Eclipse
  - memory settings [4](#)
  - Windows Vista [5](#)
- exporting
  - SAS Java projects as jars [53](#)
  - SAS Web applications as a WAR file [54](#)

## I

- import statements
  - adding [69](#)
  - Organize SAS Imports action [69](#)
  - Repository Quick Fixes [69](#)
- input parameters [33, 34](#)
- installation [3](#)
  - Eclipse 3.4 [1](#)
  - post-install [4](#)
  - Web Tools Platform [2](#)

## J

- jars
  - adding dependent [61](#)
  - adding to classpath [61](#)
  - changing order on classpath [60](#)
  - finding a class in [65](#)
  - identifying dependencies [59](#)
  - listing classpath jars [63](#)
  - listing relationships [64](#)
  - removing from classpath [59](#)
  - removing version restrictions [63](#)
  - specifying current version [62](#)
  - specifying version [62](#)
- Javadoc [68](#)
  - configuring with Eclipse [68](#)

**L**

launch files [46](#)

**M**

memory

    Eclipse settings [4](#)

Metadata Server Connection profiles

*See* [Connection profiles](#)

migrating [8](#)

model/viewer connections [70](#), [71](#)

    supported viewers [70](#)

**N**

new features [7](#)

**O**

omr.properties [46](#)

Organize SAS Imports action [69](#)

output parameters [35](#)

**P**

portlet editor [42](#)

post-install [4](#)

projects

    opening and closing [10](#)

**Q**

Quick Assists

    enabling [70](#)

**R**

requirements

Java [2](#)

SAS software [2](#)

**S**

SAS Repository [57](#)

    opening [58](#)

SAS Web applications [9](#)

    creating without Web Infrastructure

        Platform [75](#)

snippets [72](#)

StoredProcessConnection.java [47](#), [49](#)

StoredProcessDriver.java [47](#)

StoredProcessDriver.properties [48](#)

StoredProcessDriverServlet.java [48](#)

StoredProcessFacade.java [48](#), [49](#)

**T**

templates [11](#)

Tomcat [49](#)

    shutdown issue [50](#)

**V**

Versioned Jar Repository [57](#), [75](#)

**W**

Web application projects [9](#)

Web application templates [11](#)

Web Tools Platform

    installation [2](#)

Windows Vista

    Eclipse settings [5](#)

---

## Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **`yourturn@sas.com`**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **`suggest@sas.com`**.



# SAS® Publishing Delivers!

Whether you are new to the work force or an experienced professional, you need to distinguish yourself in this rapidly changing and competitive job market. SAS® Publishing provides you with a wide range of resources to help you set yourself apart. Visit us online at [support.sas.com/bookstore](http://support.sas.com/bookstore).

## SAS® Press

Need to learn the basics? Struggling with a programming problem? You'll find the expert answers that you need in example-rich books from SAS Press. Written by experienced SAS professionals from around the world, SAS Press books deliver real-world insights on a broad range of topics for all skill levels.

**[support.sas.com/saspress](http://support.sas.com/saspress)**

## SAS® Documentation

To successfully implement applications using SAS software, companies in every industry and on every continent all turn to the one source for accurate, timely, and reliable information: SAS documentation. We currently produce the following types of reference documentation to improve your work experience:

- Online help that is built into the software.
- Tutorials that are integrated into the product.
- Reference documentation delivered in HTML and PDF – **free** on the Web.
- Hard-copy books.

**[support.sas.com/publishing](http://support.sas.com/publishing)**

## SAS® Publishing News

Subscribe to SAS Publishing News to receive up-to-date information about all new SAS titles, author podcasts, and new Web site features via e-mail. Complete instructions on how to subscribe, as well as access to past issues, are available at our Web site.

**[support.sas.com/spn](http://support.sas.com/spn)**



**THE  
POWER  
TO KNOW®**