



THE
POWER
TO KNOW®



SAS® AppDev Studio™ 3.2 Eclipse Plug-ins

User's Guide

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2007. *SAS® AppDev Studio™ 3.2 Eclipse Plug-ins: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® AppDev Studio™ 3.2 Eclipse Plug-ins: User's Guide

Copyright © 2007, SAS Institute Inc., Cary, NC, USA

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

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.

March 2007

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/pubs** 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

PART 1 Usage 1

Chapter 1 △ Creating, Importing, Exporting, and Managing Projects 3

- The SAS AppDev Studio 3.2 Eclipse Plug-ins 3
- Creating a SAS Web Application Project 4
- Creating a SAS Java Project 5
- Importing a webAF 3.x Project 6
- Exporting a SAS Java Project as a Set of Jars 9
- Exporting a SAS Web Application Project as a WAR File 10

Chapter 2 △ SAS Web Application Templates 13

- The SAS Web Application Templates 13
- Using the SAS Web Application Templates 13
- Adding a Template to an Existing Project 14
- Adding Security Policies to the Web Application Server Policy File 15
- Configuring the Web Application Server Launch Settings 16
- Running the SAS Web Application 17

Chapter 3 △ Template Walk-Throughs 19

- Template Walk-Throughs Introduction 19
- Creating a SAS ID Portal Portlet 20
- Creating a SAS Management Console Plug-in 23
- Creating a SAS Data Integration Studio Plug-in 27
- Creating a DataBean 29
- Creating a JDBC Servlet 33
- Using the Information Map Servlet Templates 38
- Using the Report Viewer Servlet Template 45
- Using the Context Listener Templates 51
- Using the Logging Configuration Template 56

Chapter 4 △ Using the SAS Editor Extensions 59

- Introduction to SAS Editor Extensions 59
- Accessing SAS Component API Documentation 59
- Adding Missing Import Statements 60
- Attaching a SAS Model to a Viewer 61
- SAS Snippets 64

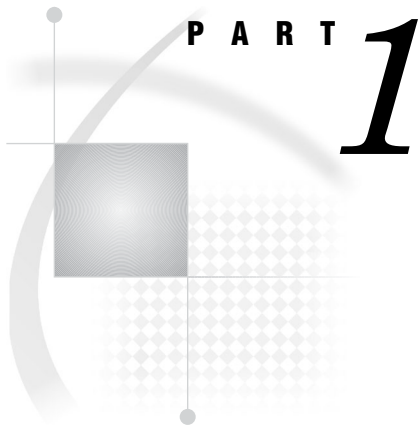
Chapter 5 △ Manual Operations for a Project's SAS Repository 67

- Overview of Manual Operations for a Project's SAS Repository 67
- Identifying Dependent Jars 68
- Removing Jars from the Classpath 69
- Changing the Order of Jars in the Classpath 69

Adding New Jars to the Classpath	69
Adding Dependent Jars	70
Specifying the Current Versions of Jars	70
Removing Version Restrictions	71
Selecting Jar Versions	71
Listing the Classpath Jars	72
Listing Jar Relationships	72
Finding a Class in a Jar	73
Changing Default Classes for SAS Java Projects	74

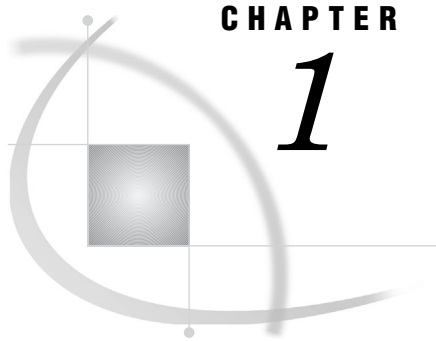
PART 2 Appendices 75

Appendix 1 △ Additional Tasks and Resources	77
Understanding SAS Web Application Projects	77
Restoring SAS Versioned Jar Repository Dependencies to SAS Web Application Projects	78
Standard SAS Filter and Servlet Declarations and Mappings	79
Importing the ADS Local Services into a SAS Metadata Repository	80
Modifying the ADS Local Services	81
Creating a New Target Run Time	82
SAS Java Web Application Resources	82
Setting up Trusted Web Authentication for SAS Web Applications	86
Specifying Separate Java Source and Output Directories	89
Index	91



Usage

<i>Chapter 1</i>	Creating, Importing, Exporting, and Managing Projects	3
<i>Chapter 2</i>	SAS Web Application Templates	13
<i>Chapter 3</i>	Template Walk-Throughs	19
<i>Chapter 4</i>	Using the SAS Editor Extensions	59
<i>Chapter 5</i>	Manual Operations for a Project's SAS Repository	67



CHAPTER

1

Creating, Importing, Exporting, and Managing Projects

<i>The SAS AppDev Studio 3.2 Eclipse Plug-ins</i>	3
<i>Creating a SAS Web Application Project</i>	4
<i>Creating a SAS Java Project</i>	5
<i>Importing a webAF 3.x Project</i>	6
<i>Importing a webAF 3.x Java Project</i>	7
<i>Importing a webAF 3.x Java Web Application Project</i>	8
<i>Upgrading a Standard Eclipse Project to a SAS Project</i>	8
<i>Running an Imported Web Application Project</i>	8
<i>Exporting a SAS Java Project as a Set of Jars</i>	9
<i>Copying the SAS Versioned Jar Repository</i>	10
<i>Using Deployment Descriptor Files</i>	10
<i>Exporting a SAS Web Application Project as a WAR File</i>	10

The SAS AppDev Studio 3.2 Eclipse Plug-ins

The SAS AppDev Studio 3.2 Eclipse plug-ins are designed to support SAS applications developers who use the open source Eclipse IDE or a third-party IDE based on the Eclipse platform. Fundamental to this support is the SAS Versioned Jar Repository and two new project types, the SAS Java Project and the SAS Web Application Project. These two new project types support access to the SAS Versioned Jar Repository.

The *SAS Versioned Jar Repository* is a collection of jars that SAS AppDev Studio uses to provide to each SAS Java Project or SAS Web Application Project the resources that each project needs. A project communicates with the SAS Versioned Jar Repository via the project's SAS Repository. A project's *SAS Repository* contains references to the specific jars required from the SAS Versioned Jar Repository.

When you use these new project types, jar dependencies are tracked for you and the required jar references are automatically added to the project as you add features to your application that rely on underlying SAS jars. You can also configure a project's SAS Repository and bypass the automation and specify which jars to include. You can also specify the version of each included jar. This flexibility enables you to manage jar dependencies on a per-project basis without disrupting past development, and contributes to a smaller distribution footprint.

SAS AppDev Studio also provides templates that assist with the development of a variety of other applications and Web applications. These applications include Portlets, SAS Management Console Plug-ins, SAS Data Integration Studio Plug-ins, web-based reporting, and OLAP solutions.

Existing SAS AppDev Studio 3.x projects can be imported and converted to SAS AppDev Studio 3.2 projects. You can also export a SAS AppDev Studio 3.2 project as a jar file, with a choice of execution modes, including the ability to launch the application using a SAS Versioned Jar Repository.

The SAS AppDev Studio 3.2 Eclipse plug-ins also provide enhancements to the standard Eclipse Java editor. These enhancements include the integration of the SAS component API documentation with the Eclipse help system, the SAS Organize Imports action, the Import from Repository Quick Fix, and a Quick Assist to support model/viewer connections.

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

Creating a SAS Web Application Project

SAS Web Application Projects are created using the Dynamic Web Project support in the Eclipse Web Tools Platform. For more information about SAS Web Application Projects, see “Understanding SAS Web Application Projects” on page 77. For more information on Eclipse Dynamic Web Projects, see “Dynamic Web Projects and Applications” at <http://help.eclipse.org/help31>.

The steps below assume you have completed the following tasks:

- configured a Tomcat 4.1.18 Web server that supports the Servlet 2.3 specification level. See “Creating a New Target Run Time” on page 82.
- ensured that the Eclipse proxy server is configured so that the Eclipse tools can navigate through your firewall to retrieve from java.sun.com any required DTDs for the **web.xml** files.

To access the proxy settings, select **Window ► Preferences** and then navigate to **Internet ► Proxy Settings**.

To create a SAS Web Application Project, follow these steps:

- 1 From the Eclipse main menu, select **File ► New ► Project**.
- 2 Expand the **SAS AppDev Studio** folder.
- 3 Select **SAS Web Application Project**.
- 4 Click **Next**.
- 5 Enter a **Project name**.

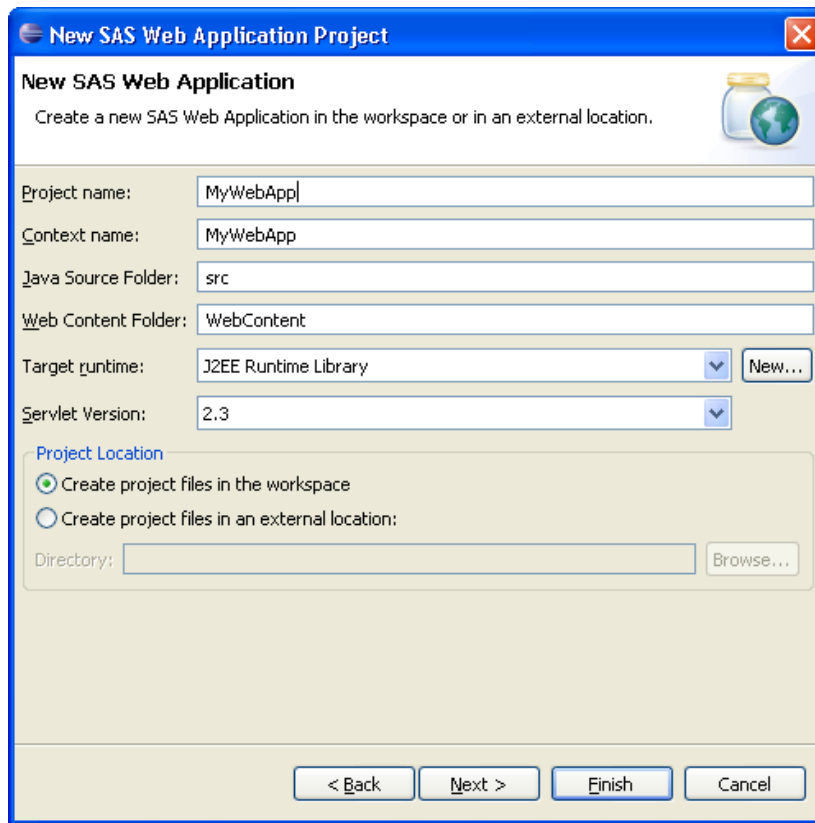
This name will be used as the context name. This name should not contain spaces.

- 6 Change the **Java Source Folder** and **Web Content Folder**, if necessary.
- 7 Select the **Target runtime** and **Servlet Version**.

The default values are *SAS local Tomcat v4.1* and *servlet version 2.3*.

- 8 Choose whether to create the project in the workspace or at an external location.

If you choose an external location, enter a path that does not contain spaces.



9 Click *Next*.

The project is created.

As part of project creation, a License Agreement dialog box for a DTD or schema might appear. Click **I Agree** to accept the license.

The new project will reference the standard set of SAS Versioned Jar Repository files for SAS Web applications and the associated static content from the SAS Web Infrastructure Kit. For details, see “Understanding SAS Web Application Projects” on page 77.

10 If you want to use a template, select the **Add Template Content check box.**

Templates can be used to create the initial code in your project. For information about working with templates, see “Using the SAS Web Application Templates” on page 13.

Creating a SAS Web Application Project without a template results in a project that lacks the standard servlet mappings. For more information, see “Standard SAS Filter and Servlet Declarations and Mappings” on page 79.

Creating a SAS Java Project

When you create a SAS Java Project in Eclipse, references to the required jars in the SAS Versioned Jar Repository are automatically added to the project.

To create a SAS Java Project, follow these steps:

- 1 From the Eclipse main menu, select **File ► New ► Project**.
- 2 Expand the **SAS AppDev Studio** folder.

- 3 Select **SAS Java Project**.
- 4 Click **Next**.
- 5 Enter a **Project name**, set the **JDK Compliance** to 1.4, and select the appropriate **Project layout** option.
- 6 Click **Next**.

The Java Settings appear.

- 7 Select the **Libraries** tab.

The **Libraries** tab displays the jars available to the project, including those jars in the SAS Versioned Jar Repository. Edit the list according to your needs.

- 8 Click **Finish**.
- 9 If you are asked to switch to the Java perspective, click **Yes**.

For more information about configuring the SAS Versioned Jar Repository, see Chapter 5, “Manual Operations for a Project’s SAS Repository,” on page 67.

For details about working with templates, see “Using the SAS Web Application Templates” on page 13.

Importing a webAF 3.x Project

You can import a SAS AppDev Studio 3.x Java or Java Web Application project and convert it to a SAS Java or SAS Web Tools project compatible with AppDev Studio 3.2 and Eclipse. You do not need webAF 3.x software installed on your development machine to perform the import; you only need access to the project files.

For all imported projects, SAS libraries are updated to use SAS AppDev Studio 3.2 classes provided through the SAS Versioned Jar Repository classpath container.

If you are importing a Web application project, these changes are also applied to the project:

- SAS static content files are replaced with new versions.
- The **java.webapp.policy** file is replaced with an updated file containing new security settings.

Before importing a Web application project, you must ensure that the proxy server is configured so that the Eclipse tools can navigate through your firewall to retrieve from java.sun.com any required DTDs for the **web.xml** files.

To access the proxy settings, select **Window ► Preferences** and then navigate to **Internet ► Proxy Settings**.

To import a webAF 3.x project into Eclipse, follow these steps:

- 1 Select **File ► Import**.
- The Import dialog box appears.
- 2 Select **webAF Project Import**.
- 3 Click **Next**.

The webAF Project Import dialog box appears.

If webAF 3.x is installed, the **Root Directory** field is populated with the location of the webAF projects directory, and a list of projects from the webAF projects directory is provided. Use the **Browse** button to choose an alternate location.

If webAF 3.x is not installed, you must specify the **Root Directory** of the project that you want to import.

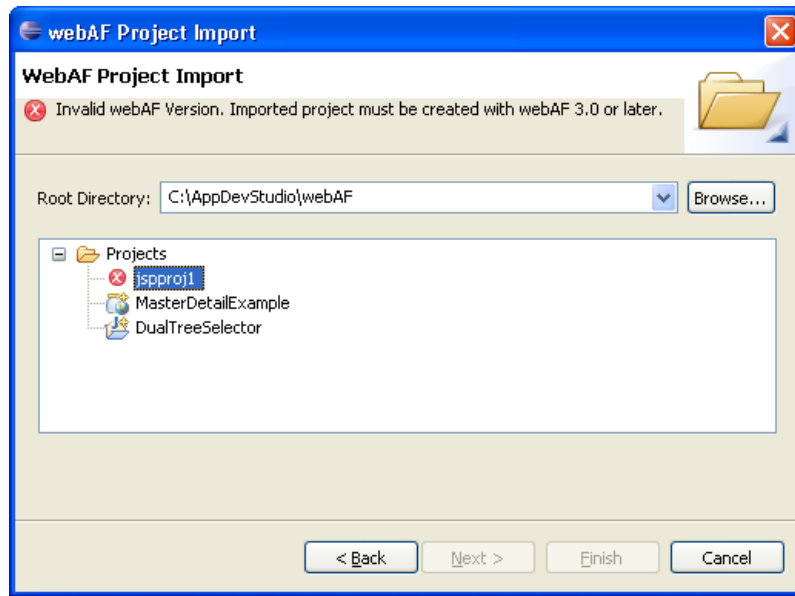
4 Select the project that you want to import.

Projects marked with a *J* icon are Java projects.

Projects marked with a jar icon are Web Application projects.

Projects marked with a red X icon are projects that cannot be imported. Projects cannot be imported if any of the following are true:

- ☐ The project is not a valid webAF project or does not contain all the required project files.
- ☐ The project is from a version of webAF older than 3.0. To upgrade the project to 3.0, install webAF 3.0, open the project, and then save it.



5 Click **Next**.

If you selected a Java Project, follow the steps in “Importing a webAF 3.x Java Project” on page 7.

If you selected a Web Application Project, follow the steps in “Importing a webAF 3.x Java Web Application Project” on page 8.

Importing a webAF 3.x Java Project

These instructions assume that you have selected a Java Project in “Importing a webAF 3.x Project” on page 6.

To continue importing a webAF 3.x Java Project, follow these steps:

- 1 Specify a name for the project in the **Project name** field.
- 2 Specify the **JDK Compliance**. For SAS applications this should be 1.4.
- 3 Specify the **Project layout**.
- 4 Click **Finish**.

The project is imported into the new Eclipse workspace and opened using the Java perspective. The new project is based on a SAS Java Project, and is configured with the default set of jars needed for most SAS Java Projects.

If you have the Eclipse autobuild option enabled, then the project will build automatically. Otherwise, you must manually build the project.

If any errors occur due to missing import statements, use the SAS Organize Imports action to add the necessary import statements and any required jars to the project's

SAS repository (see “Adding Missing Import Statements” on page 60). If additional jars are required to build and execute your project, add them as described in Chapter 5, “Manual Operations for a Project’s SAS Repository,” on page 67. You can also add jars to the classpath using the normal Eclipse mechanism.

When you import a project, a launch configuration is automatically created. To run the project, right-click the project’s Main class in the Eclipse Package Explorer view, and select **Run As ► SAS Java Application**.

Importing a webAF 3.x Java Web Application Project

These instructions assume that you have selected a SAS Web Application Project in “Importing a webAF 3.x Project” on page 6.

To continue importing a Java Web Application Project from webAF 3.x, follow these steps:

- 1 For **Target runtime**, select **SAS local Tomcat v4.1**.
- 2 Do not alter the defaults for any other field.
- 3 Click **Finish**.

The project is imported into the new Eclipse workspace and opened using the Java perspective. The new project is based on a SAS Java Web Application Project, and is configured with the default set of jars needed for most SAS Java Web Application Projects.

If you have autobuild enabled, then the project will build automatically. Otherwise, you must manually build the project.

The imported project is structured as a standard Eclipse Web Tools Platform Dynamic Web Project. Java source code is located in the **/src** folder. All other content is located in the **/WebContent** folder with the normal WAR file structure.

Upgrading a Standard Eclipse Project to a SAS Project

You can also upgrade standard Eclipse Java projects or Dynamic Web projects to their SAS counterparts.

To upgrade a project, follow these steps:

- 1 Right-click the project and select **Properties**.
- 2 Select **SAS Java Project Properties**.
- 3 Select the **Upgrade to SAS Java Project** check box.
- 4 Click **OK**.

Running an Imported Web Application Project

Running an imported Web application project involves the following steps:

- ☐ Configuring the server to serve the Web application.
- ☐ Starting the server.
- ☐ Entering the appropriate URL for the resource you want to view.

To configure the server, follow the procedures found in “Adding Security Policies to the Web Application Server Policy File” on page 15 and “Configuring the Web Application Server Launch Settings” on page 16.

To display the resource you want to view, follow the procedure in “Running the SAS Web Application” on page 17.

If the imported SAS Web Application Project contained a webAF Web Application example template, the resource to display for that example is its controller servlet.

There are no special server requirements for running an imported webAF 3.x Java project.

Exporting a SAS Java Project 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[])**, and are thus true applications.

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. This directory must already exist.

7 Specify the **Jar Name**.

8 Click **Next**.

9 Select the **Use SAS Repository** check box if you want to create a jar that runs against an installed copy of the SAS Versioned Jar Repository.

If you choose not to deploy against the repository, then all the required jars are copied into the deployment directory.

If you choose to deploy against a SAS Versioned Jar Repository, select the **SAS repository locator** check box to include in the created jar the manifest entries that allow your application to be run with **java -jar <jarname>**.

If you choose not to run with the SAS repository locator, then a batch file is created. You must edit the batch file and specify the location of the **sas.app.launcher** jar.

10 Select the **Main class** that starts your application. The main class must reside in your project.

11 Click **Next**.

12 Select the **Save Deployment Descriptor in Project** check box if you want to capture the current export settings. For more information on descriptor files, see “Using Deployment Descriptor Files” on page 10.

If you choose this option, specify the path to the descriptor file, relative to the project, in the **Descriptor Location** field. The filename must end in **.depdesc**.

13 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.

14 Click **Finish**.

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

Copying the SAS Versioned Jar Repository

The SAS Versioned Jar Repository was installed when SAS AppDev Studio 3.2 was installed. There is no separate installer for the SAS Versioned Jar Repository. If you export a project that uses jars in the SAS Versioned Jar Repository, you must copy the SAS Versioned Jar Repository to the target machine. For information on exporting an application for distribution, see “Exporting a SAS Java Project as a Set of Jars” on page 9.

Follow these steps to copy the repository to another machine:

- 1 Create a zip file of the `\VersionedJarRepository\` directory.

This directory is usually located in

`C:\Program Files\SAS\SASAppDevStudio\3.2\`.

- 2 Move the zip file to the target machine and unzip it.
- 3 Open a command prompt and navigate to `\VersionedJarRepository\eclipse\plugins\`.
- 4 Issue the following command:

```
java -jar sas.app.launcher_3.2.0.jar -install
```

You must be an administrator on Windows or a superuser on Linux.

Using Deployment Descriptor Files

After you have exported a project and saved the export settings to a `.depdesc` file, you can export the project again without going through the Export dialog box.

Once you have a descriptor file in a project, the settings for the Export dialog box are read from it. You can have multiple descriptor files per project. Unless otherwise specified, the first descriptor file listed is used to populate the Export dialog box.

To run a `.depdesc` file without using 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 run a `.depdesc` file using the Export dialog box, right-click the `.depdesc` file and select **Open SAS Export**.

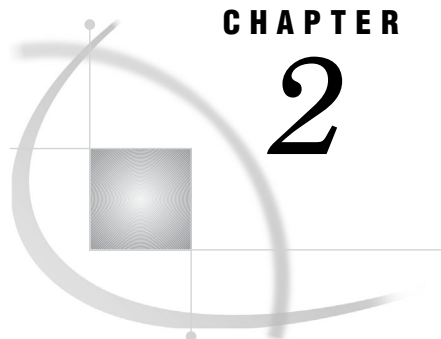
To inspect or change the settings in a descriptor file, you can open the file by double-clicking it, and editing values as needed. The Save command is available only after you change an option and then click on a different line.

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 Select **File ► Export**.
- 2 Select **WAR file** as the export destination, and then click **Next**.
- 3 Select the **Web module** name that matches the name of the project.
- 4 Specify the **Destination** file for the WAR file.
- 5 Select the **Export source files** and **Overwrite existing file** check boxes as needed.
- 6 Click **Finish**.

For details about deploying the WAR file, see *Deploying Web Applications* on the AppDev Studio Developer's Site at <http://support.sas.com/rnd/appdev/doc/DeployingWebApps.html>.



CHAPTER

2

SAS Web Application Templates

<i>The SAS Web Application Templates</i>	13
<i>Using the SAS Web Application Templates</i>	13
<i>Adding a Template to an Existing Project</i>	14
<i>Adding Security Policies to the Web Application Server Policy File</i>	15
<i>Configuring the Web Application Server Launch Settings</i>	16
<i>Running the SAS Web Application</i>	17

The SAS Web Application Templates

SAS Web Application templates are collections of starter code to support rapid development of SAS Web applications. You can add these templates to a project when it is created, or later, to an existing SAS Web Application project or Web Tools Dynamic Web project. If you add a template to an existing Web Tools Dynamic Web project, the project is converted to a SAS Web Application Project.

Using the SAS Web Application Templates

Several of the SAS Web Application templates use the resources of one or more servers (for example, a SAS Workspace Server or SAS IOM Spawner). For each server that uses the Foundation Services, the host specified in the template must match the host configured in the SAS Metadata Repository. For example, the SAS Workspace Server host specified in the template must be the same host as the SAS Workspace Server specified in the repository. The host for such a server, and possibly its port number, can be specified when the template is added to the project. The default value for the host will be the machine name of the system running SAS AppDev Studio 3.2 Eclipse Plug-ins.

In addition to matching server information, several of the SAS Web Application templates use the ADS Local Services that are deployed from a SAS Metadata Repository. You must ensure that the ADS Local Services are available by importing them into your SAS Metadata Repository using the SAS Management Console. See “Importing the ADS Local Services into a SAS Metadata Repository” on page 80.

Finally, if the SAS Metadata Repository is on a system other than the one on which SAS AppDev Studio 3.2 Eclipse Plug-ins was installed, the SAS Metadata Repository will require modification. See “Modifying the ADS Local Services” on page 81.

For information about SAS Web Application Projects, see “Understanding SAS Web Application Projects” on page 77.

Adding a Template to an Existing Project

To begin the process of adding a template to an existing project, follow these steps:

- 1 From the Eclipse main menu, select **File ► New ► Other**.
- 2 Expand **SAS AppDev Studio**.
- 3 Select **Add Template Content to Project**.
- 4 Click **Next**.
- 5 Select the project to which you want to add a template.
- 6 Select the filtering for the templates.
- 7 The **Template** box lists the templates available to the project, according to the selected filter.

The SAS DataBean Wizard category contains a template that can be used with SAS Web Application projects or SAS Java Projects. This template works with both SAS Java Projects and SAS Web Application Projects. For more information, see “Creating a DataBean” on page 29.

All the other templates are specific to SAS Web Application Projects and are organized into two categories: Foundation Services Support, and SAS Web Application Examples.

The Foundation Services Support templates provide Java files and other resources useful to Web applications based on the SAS Foundation Services. For details about these templates, see the AppDev Studio 3.2 Developer’s Site: <http://support.sas.com/rnd/appdev/index32.html>.

The SAS Web Application Examples templates provide resources in various states of completion. The amount of user modification required to bring the example to a functional state varies depending on the template. The *Model 2 Servlet* template provides only the simplest code, so the majority of the implementation is left to you. At the other end of the spectrum, the Information Map Viewer Servlet and Report Viewer Servlet templates create examples that are ready to run using only the information provided when you add the template.

Some templates that require only a small amount of user modification display an item in the Eclipse Problems view or Tasks view to indicate where modification is needed. For details about these templates, see the AppDev Studio 3.2 Developer’s Site: <http://support.sas.com/rnd/appdev/index32.html>.

- 8 Select the template to add to the project.

If you are adding a template to a Dynamic Web Project, an error might appear stating that the selected project does not meet the template’s requirements. If the **Upgrade Project** button is enabled, you can click it to upgrade the project to meet the template’s requirements.

CAUTION:

If you are adding a **SAS JDBC DataBean Class** template to a **Dynamic Web Project**, do not upgrade the project. Doing so results in an incomplete project. Instead, select any of the SAS Web Application templates and click the **Upgrade Project** button. After the project is upgraded, you can then add the SAS JDBC DataBean Class template. △

- 9 Click **Next** and enter any information specific to the template you chose.
- 10 Several of the templates require additional Tomcat server configuration before the SAS Web Application project can run on the server:

- a The launch configuration for the Tomcat server must be updated to include additional JVM options and system properties. See “Configuring the Web Application Server Launch Settings” on page 16.
- b The **catalina.policy** file must be updated to add permissions specific to the Web application. See “Adding Security Policies to the Web Application Server Policy File” on page 15.

Adding Security Policies to the Web Application Server Policy File

SAS Web Application Projects are intended to be run from a server with a security manager. Thus, appropriate permissions must be granted to the Web application so that it can run. The SAS Web Application templates include a **java.webapp.policy** file in the root of the SAS Web Application Project that contains these necessary permissions.

The **java.webapp.policy** file is a full Tomcat policy file which could be used as a replacement for the Tomcat **catalina.policy** file if this project was the only project served. However, because a server almost always serves more than one application, the procedure below better supports running multiple SAS Web Application Projects on the same Tomcat server by copying the permissions from the **java.webapp.policy** file to the Tomcat **catalina.policy** file.

The following instructions assume you are in the J2EE perspective, and that the **catalina.policy** file has never been updated before. Do not include the permissions from a project in the **catalina.policy** file more than once.

To update the server’s policy file, you must paste content from two places in the **java.webapp.policy** file, into the server policy file **catalina.policy**. Follow these steps:

- 1 In the Project Explorer, expand **Other Projects**.
- 2 Expand **Servers**, and then **Tomcat v4.1 Server @ localhost-config**.
- 3 Double-click the **catalina.policy** file under the **Tomcat v4.1 Server @ localhost-config**.
- 4 In the Project Explorer, expand **Dynamic Web Projects**.
- 5 Expand the project you are interested in and double-click the **java.webapp.policy** file found in the project’s root folder.
- 6 In the **java.webapp.policy** file, locate the following comment:

```
// ===== <Your Project Name> Web Application Permissions =====
```

- 7 Copy to the clipboard that comment line to the end of the file (approximately 100 lines).
- 8 Scroll to the bottom of the **catalina.policy** file and paste the clipboard contents from the **java.webapp.policy** file.
- 9 In the **java.webapp.policy** file, locate the two comments that begin with **NEW**:

```
// NEW: Addition permissions not in Tomcat distribution
and
```

```
// NEW: Allow access to default parser and transformer
```

- 10 Copy the NEW comment lines and their associated permissions. Do not to copy the closing bracket at the end of the permissions.
- 11 Paste the NEW comments and permissions into the WEB APPLICATION PERMISSIONS section of the **catalina.policy** file (inside the *grant { }* segment). This section defines the permissions granted to all Web applications.

```
// ===== WEB APPLICATION PERMISSIONS =====

// These permissions are granted by default to all Web applications
// In addition, a Web application will be given a read FilePermission
// and JndiPermission for all files and directories in its document root.
grant {
    // Required for JNDI lookup of named JDBC DataSource's and
    // javamail named MimePart DataSource used to send mail
    permission java.util.PropertyPermission "java.home", "read";
    permission java.util.PropertyPermission "java.naming.*", "read";
    permission java.util.PropertyPermission "javax.sql.*", "read";
    .
    .
    .
};
```

12 Save and close the **catalina.policy** file.

13 Close the **java.webapp.policy** file.

The system properties in the SAS AppDev Studio Web Application Permissions section are documented in the comments. All but one property specifies a host or port for the needed servers. The one exception is a system property that specifies the name under which the Web application will be served.

Configuring the Web Application Server Launch Settings

To support SAS Web application requirements, you must set Java VM options on the Web application server. For example, SAS Web Application Projects require heap sizes greater than what the Java VM provides by default.

The SAS Web Application templates provide the necessary Java VM arguments in a **launchParameters.txt** file. To add these arguments to the launch configuration used by the Tomcat server you must copy lines from the **launchParameters.txt** file into the **VM Arguments** field in the Eclipse interface. The following procedure assumes you are in the J2EE perspective.

To update the server's launch configuration file, follow these steps:

- 1 In the Project Explorer, expand **Dynamic Web Projects**.
- 2 Expand the project you are interested in and double-click the **launchParameters.txt** file.

This file is located in the project, and contains declarations for system properties and VM arguments that must be added as Java VM arguments to the server's run configuration to specify how the server is started.

- 3 Remove the carriage returns between those system properties lines.

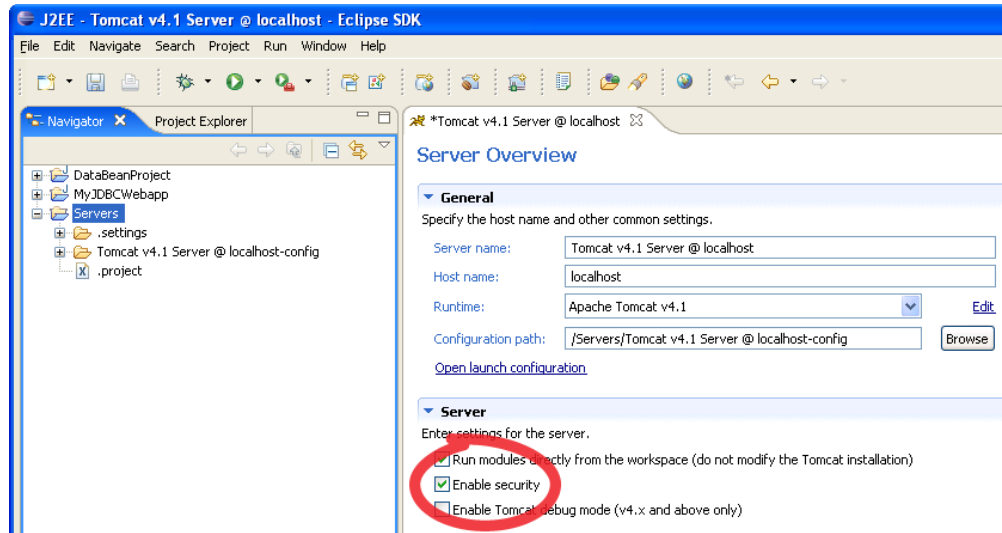
The lines that contain properties must be merged into a single line before being inserted into the **VM Arguments** field in the next step. It is easier to remove the carriage returns before, rather than after, the arguments are pasted. If you fail to remove a line end, the launch configuration will malfunction.

For example, these are the arguments generated in the **launchParameters.txt** when you add an Information Map Viewer Servlet template to your project:

```
-DsaswebApp2.name=saswebApp2
-Xmx256m -Xms128m
-Djava.security.auth.login.config="C:\Documents and Settings\user\
workspace\saswebApp2\login.config"
```

```
-Dsas.omr.host=omr.server.host
-Dsas.omr.port=8561
-Dsas.workspace.host=workspace.server.host
-Dsas.olap.host=sasOlap.server.host
-Dsas.stp.host=stp.server.host
```

- 4 Copy to the clipboard the line that now contains all the arguments. Do not include a trailing carriage return.
- 5 In the Servers view, right-click on the **Tomcat v4.1 Server @ localhost** and select **Open**. This opens this server's configuration in the server editor.
- 6 Verify that the **Enable security** check box is selected.



- 7 Click the **Open launch configuration** link.
- 8 In the launch configuration dialog box, select the **Arguments** tab.
- 9 Scroll to the bottom of the **VM Arguments** field and place the cursor at the end of the existing contents.
- 10 Enter a SPACE at the end of the existing content.
- 11 Paste the text from the clipboard.
- 12 Click **OK**.
- 13 Close the server editor.

Running the SAS Web Application

After a server has been configured, running a SAS Web Application involves three steps:

- 1 The project must be added to the Tomcat server.
- 2 The server must be started.
- 3 The browser must be invoked with the proper URL.

The first two steps can be performed separately. However, the command to perform the third step also performs the first two steps automatically if they have not already occurred. The steps given below takes advantage of this. The following procedure assumes you are in the J2EE perspective.

To add the project to the server and display it, follow these steps:

- 1 Stop the *Tomcat v4.1 Server @ localhost* by switching to the Servers view in Eclipse, right-clicking on the server, and selecting **Stop**.

Stopping the server simplifies the remainder of the procedure. You can allow the server to run if all three of the following criteria are true:

- ☐ The server is already running in the desired mode of execution.
- ☐ The project has already been added to the server.
- ☐ There have been no changes to the project or only static files, such as JSP and HTML files, have been changed.

- 2 From a navigator or an editor view, right-click the resource you want to display in the browser. For example, you could choose the source file of the servlet that was added by a SAS Web Application template.

The servlet's name will end with `ControllerServlet`, and will start with the name of the SAS Web Application example. For example, **InfoMapViewExampleControllerServlet.java**. This assumes that the default name of the servlet was accepted when the template was created.

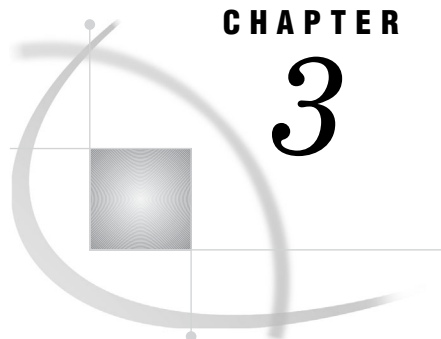
- 3 Choose the mode of execution by selecting one of the following:

- ☐ **Run As ► Run on Server**
- ☐ **Debug As ► Debug on Server**
- ☐ **Profile As ► Profile on Server**

This option is available if the Tracing and Profiling Tools are installed from the Test and Performance Tools Platform (TPTP) project.

- 4 Select the **Tomcat v4.1 Server @ localhost server** and click **Finish**.

When you click the **Finish** button, Eclipse adds the project to the server if the project was not already there. When a project is published to the server for the first time, the copying process can take a several minutes as there are many SAS jars and content files to copy. After the copying is complete, the server is started, and Eclipse tests if it can accept requests. Once the server is accepting requests, a browser is invoked with the appropriate URL for the selected resource.



CHAPTER

3

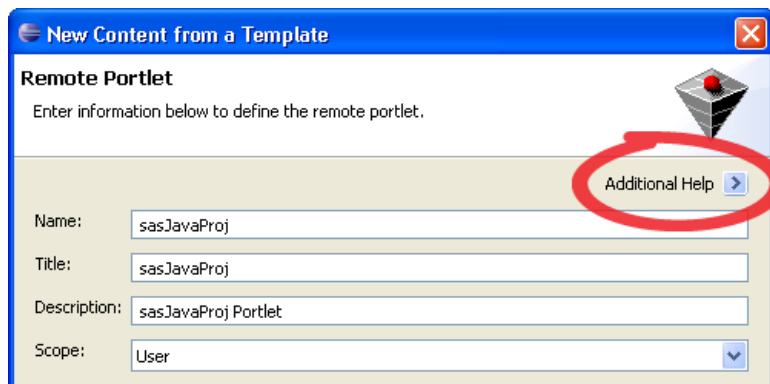
Template Walk-Throughs

<i>Template Walk-Throughs Introduction</i>	19
<i>Creating a SAS ID Portal Portlet</i>	20
<i>Packaging the Portlet into a PAR File</i>	23
<i>Creating a SAS Management Console Plug-in</i>	23
<i>Creating a SAS Data Integration Studio Plug-in</i>	27
<i>Creating a DataBean</i>	29
<i>Creating a JDBC Servlet</i>	33
<i>Completing a JDBC Servlet</i>	37
<i>Preparing the Server to Run the JDBC Servlet</i>	37
<i>Running the JDBC Servlet</i>	37
<i>Using the Information Map Servlet Templates</i>	38
<i>Completing an Information Map Viewer Servlet</i>	43
<i>Completing an Information Map TableView or OLAPTableView Servlet</i>	44
<i>Completing an Information Map Default Servlet</i>	44
<i>Preparing the Server to Run the Information Map Servlet Application</i>	44
<i>Running an Information Map Servlet Web Application</i>	45
<i>Using the Report Viewer Servlet Template</i>	45
<i>Completing a Report Viewer Servlet</i>	50
<i>Preparing the Server to Run the Report Viewer Application</i>	51
<i>Running the Report Viewer Application</i>	51
<i>Using the Context Listener Templates</i>	51
<i>Using the Logging Configuration Template</i>	56

Template Walk-Throughs Introduction

The following sections provide step-by-step walk-throughs for using the templates shipped with SAS AppDev Studio 3.2 Eclipse Plug-ins.

In addition to these walk-throughs, many of the template dialog boxes have field-sensitive online Help available. To access this help, toggle the **Additional Help** button in the upper right of the dialog box, and then click inside the field you need help on.



Creating a SAS ID Portal Portlet

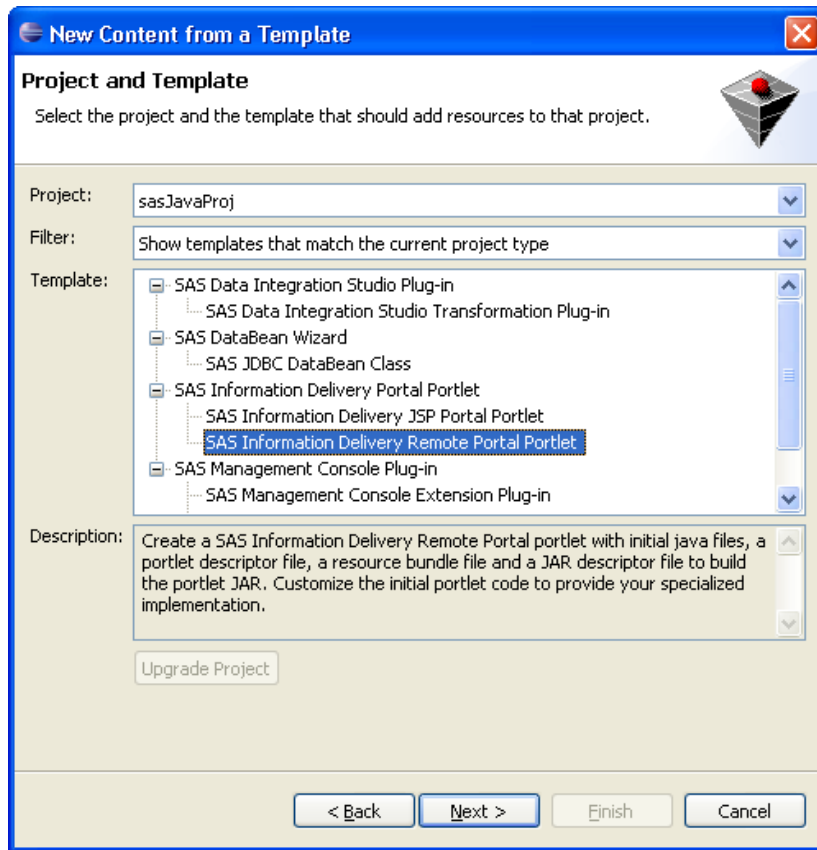
There are two ways you can create a SAS ID Portal portlet:

- Create a SAS Java Project and then add the SAS ID Portal Portlet template to it by selecting the **Add Template Content** check box at the end of the creation process.
- Create a SAS Java Project and then add the SAS ID Portal Portlet template later.

To create a SAS ID Portal Portlet from a new SAS Java Project, follow these steps:

- 1 Specify separate source and output directories (see “Specifying Separate Java Source and Output Directories” on page 89).
- 2 Create a new SAS Java Project (see “Creating a SAS Java Project” on page 5).
- 3 Select **File ► New ► Other**.
- 4 Expand **SAS AppDev Studio**.
- 5 Select **Add Template Content to Project**.
- 6 Click **Next**.
- 7 In the **Project** field, select the SAS Java Project to which you want to add the template content (the new project created in step 2).
- 8 Expand **SAS Information Delivery Portal Portlet**.
- 9 Select **SAS Information Delivery Remote Portal Portlet**.

The **Upgrade Project** button is enabled only if the project to which you are adding the portlet is not already a SAS Java Project. In such cases, clicking the **Upgrade Project** button adds to the project the SAS Jar Repository and the SAS Java Project builders.



10 Click **Next**.

11 Specify options for the portlet.

The fields here correspond to tag entries in the Portlet Deployment Descriptor DTD. For example, the following fields correspond to attributes of the same name on the <local-portlet> or <remote-portlet> element:

- ☐ Title
- ☐ Description
- ☐ Scope
- ☐ Editor Type

Detailed element descriptions for the Portlet Deployment Descriptor DTD are available at http://support.sas.com/rnd/itech/doc9/portal_dev/dtd2html/DTD-HOME.html.

New Content from a Template

Remote Portlet

Enter information below to define the remote portlet.

Additional Help >

Name: sasJavaProj

Title: sasJavaProj

Description: sasJavaProj Portlet

Scope: User

Editor type: None

Options

- ☐ Pass Context ID
- ☐ User Can Create More
- ☒ Auto Deploy
- ☐ Show Edit Properties

Files

Icon source: remote/resources/icon.gif

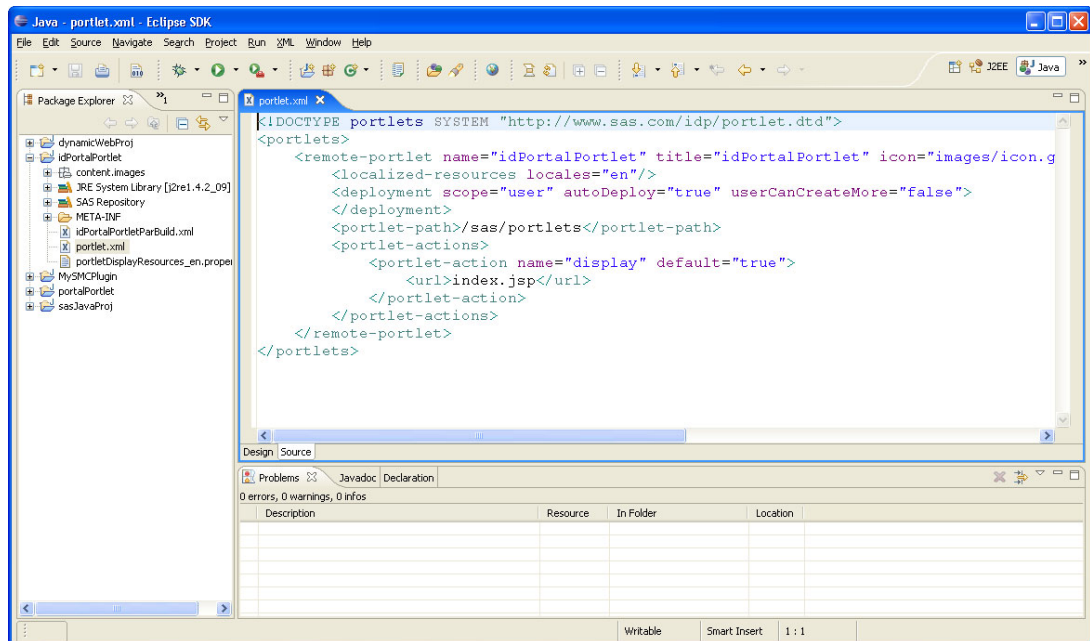
Icon filename: icon.gif

URL: index.jsp

< Back Next > Finish Cancel

12 Click *Finish*.

The basic ID Portal portlet descriptor is generated:



For information on modifying the generated portlet descriptor for your needs and deploying it, see the SAS Web Infrastructure Kit: Developer's Guide (http://support.sas.com/rnd/itech/library/toc_portaldev.html).

Packaging the Portlet into a PAR File

To package the new portlet into a PAR file, go to the top of the project directory in the Navigator view and follow these steps:

- 1 Right-click the **<portletname>.jardesc** file and select **Create PAR**.

A PAR file is created in the project directory.

- 2 Copy the PAR file to your ID Portal's deployment directory.

If this is a new portlet, the Portal should discover the portlet and allow you to access it. If you are replacing an existing portlet, the Portal must be restarted before the replacement portlet is available.

Creating a SAS Management Console Plug-in

There are two ways you can create a SAS Management Console plug-in:

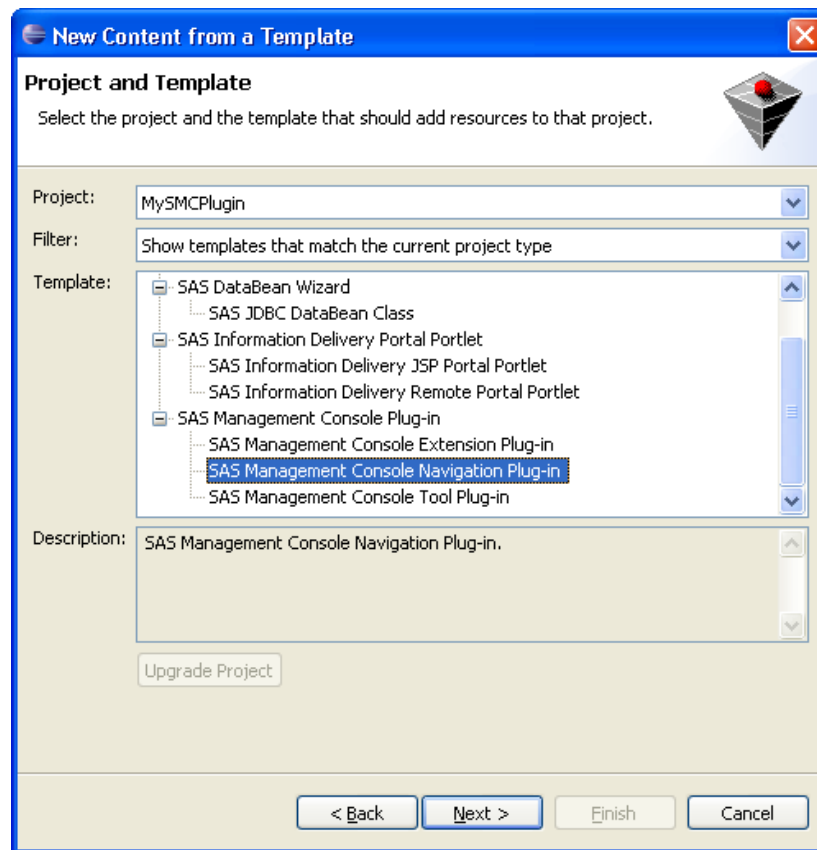
- ☐ Create a SAS Java Project and then add content to it by selecting the **Add Template Content** check box at the end of the creation process.
- ☐ Create a new SAS Java Project and then add content to it later.

To create a SAS Management Console plug-in from a new SAS Java Project, follow these steps:

- 1 Specify separate source and output directories (see "Specifying Separate Java Source and Output Directories" on page 89).
- 2 Create a new SAS Java Project (see "Creating a SAS Java Project" on page 5).
- 3 Select **File ► New ► Other**.

- 4 Expand **SAS AppDev Studio**.
- 5 Select **Add Template Content to Project**.
- 6 Click **Next**.
- 7 In the **Project** field, select the SAS Java Project to which you want to add the template content (the new project created in step 2).
- 8 Expand **SAS Management Console Plug-in**.
- 9 Select **SAS Management Console Navigation Plug-in**.

The **Upgrade Project** button is enabled only if the project to which you are adding the SAS Management Console plug-in is not already a SAS Java Project. In such cases, clicking the **Upgrade Project** button adds to the project the SAS Jar Repository and the SAS Java Project builders.



- 10 Click **Next**.
- 11 Specify the **Name**, **Version**, **Copyright**, and icon information for the SAS Management Console plug-in.

New Content from a Template

SAS Management Console Plug-in Wizard

Enter information below to define the plug-in.

Additional Help >

Name: MySMCPlugin

Version: 1.0

Copyright: 2006

Description: SAS AppDev Studio generated SAS Management Console Navigation Plug-in

Icon source: navigation/images/icon.gif Browse...

Icon filename: icon.gif

< Back Next > Finish Cancel

12 Click **Next**.

13 Specify other data and resources for the plug-in.

New Content from a Template

SAS Management Console Navigation Plug-in Wizard

Enter information below to define the plug-in.

Additional Help >

Plug-in Type: Navigation

Package: com.sas

Tooltip text: SAS AppDev Studio generated SAS Management Console Navigation Plug-in

File name prefix: MySMCPlugin

Files

Jar file name: smc.MySMCPlugin.jar Browse...

Options

☒ Require a metadata connection

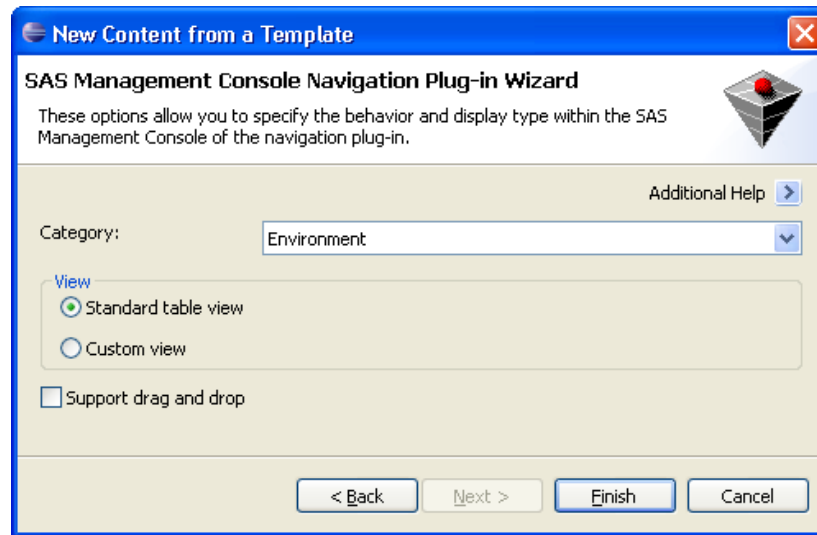
☐ Implement the PluginExtensionInterface

☐ Implement the PluginToolInterface

< Back Next > Finish Cancel

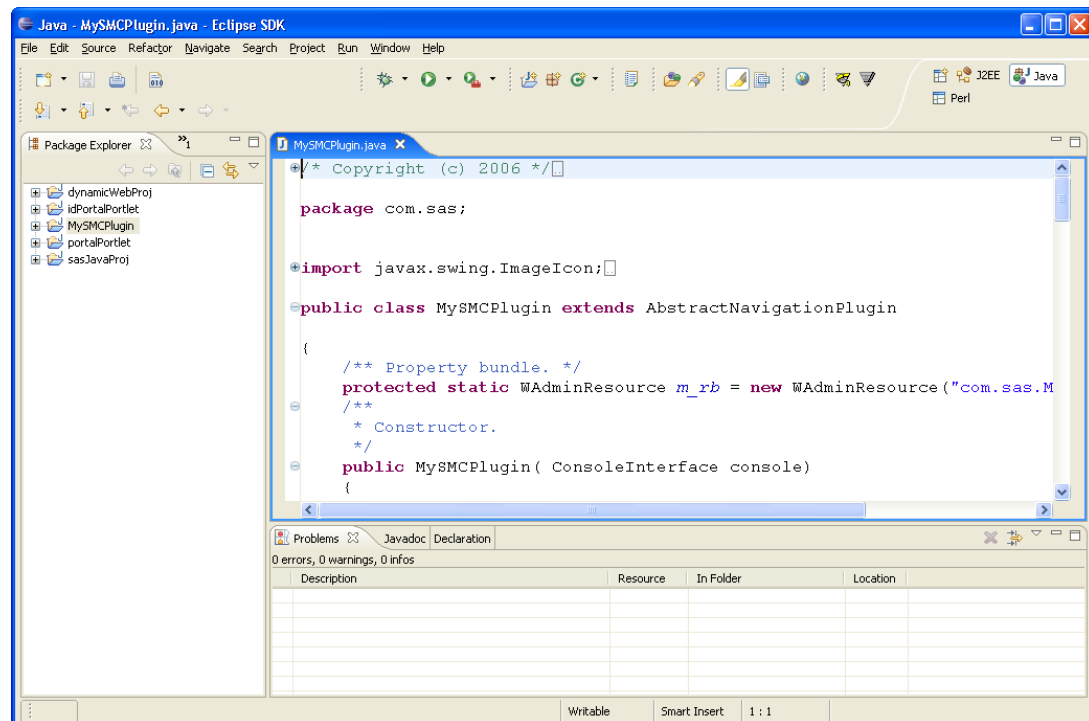
14 Click **Next**.

15 Specify the **Category**, **View**, and drag-and-drop support.



16 Click **Finish**.

The SAS Management Console plug-in code is generated given the options you selected.



Customize the generated code to suit your needs. For information on writing and deploying SAS Management Console plug-ins, see the SAS Management Console Plug-in Developer Guide, available from the **doc.zip** file that is installed with SAS Management Console (the default location is

C:\Program Files\SAS\SASManagementConsole\9.1\doc.zip).

Creating a SAS Data Integration Studio Plug-in

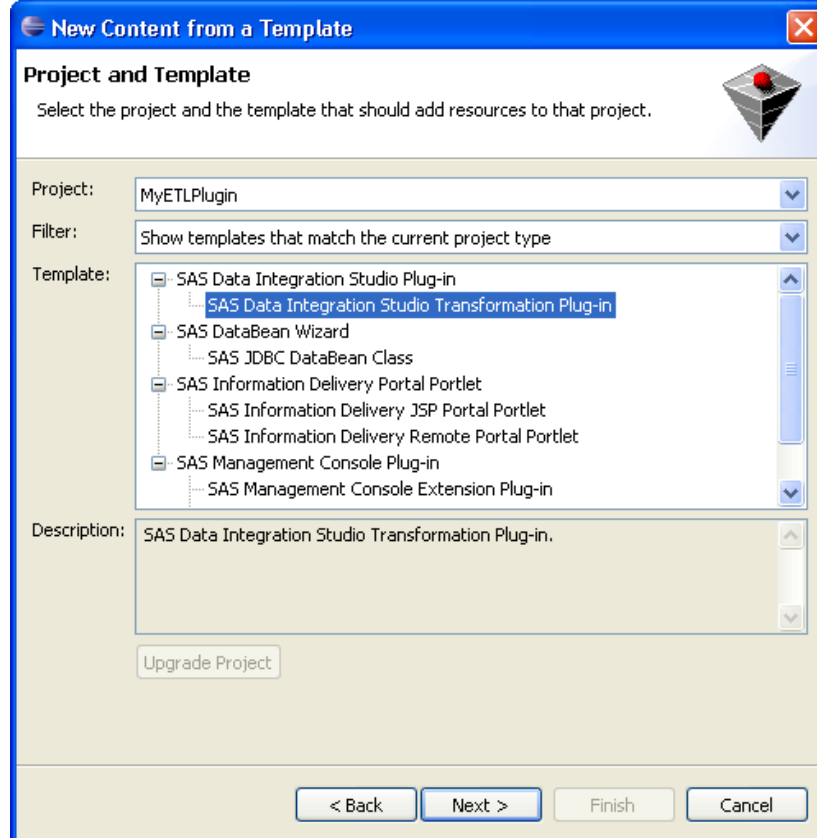
There are two ways you can create a SAS Data Integration Studio plug-in project:

- Create a SAS Java Project and then add content to it by selecting the **Add Template Content** check box at the end of the creation process.
- Create a new SAS Java Project and then add content to it later.

To create a Data Integration Studio plug-in from a new SAS Java Project, follow these steps:

- 1 Specify separate source and output directories (see “Specifying Separate Java Source and Output Directories” on page 89).
- 2 Create a new SAS Java Project (see “Creating a SAS Java Project” on page 5).
- 3 Select **File ► New ► Other**.
- 4 Expand **SAS AppDev Studio Templates**.
- 5 Select **Add Template Content to Project**.
- 6 Click **Next**.
- 7 In the **Project** field, select the SAS Java Project to which you want to add the template content (the new project created in step 2).
- 8 Expand **SAS Data Integration Studio Plug-in**.
- 9 Select **SAS Data Integration Studio Transformation Plug-in**.

The **Upgrade Project** button is enabled only if the project to which you are adding the SAS Data Integration Studio Plug-in is not a SAS Java Project. In such cases, clicking the **Upgrade Project** button adds to the project the SAS Jar Repository and the SAS Java Project builders.



10 Click **Next**.

11 Specify the **Name**, **Version**, **Copyright**, and icon information for the Data Integration Studio plug-in.

New Content from a Template

SAS Data Integration Studio Plug-in Wizard
Enter information below to define the plug-in.

Additional Help >

Name:

Version:

Copyright:

Description:

Icon source:

Icon filename:

< Back Next > Finish Cancel

12 Click **Next**.

13 Specify the Java **Package**, **File name prefix** for the generated files, and the **Jar file** to be used for packaging the plug-in.

New Content from a Template

SAS Data Integration Studio Transformation Plug-in
Define a SAS Data Integration Studio Transformation Plug-in.

Additional Help >

Package:

File name prefix:

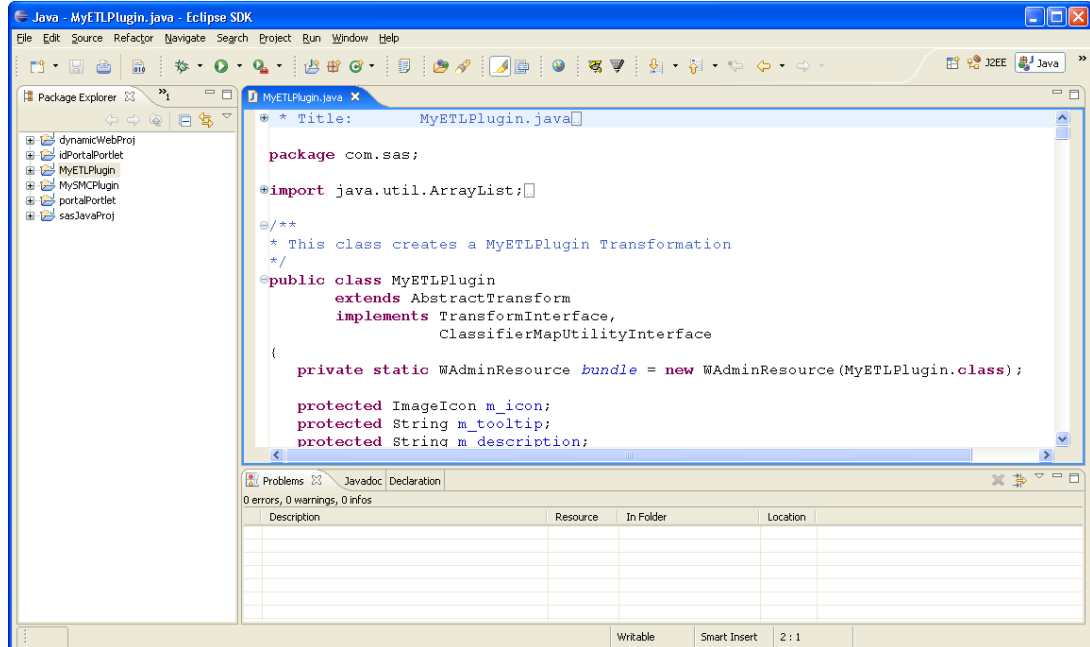
Files

Jar File:

< Back Next > Finish Cancel

14 Click Finish.

The SAS Data Integration Studio plug-in code is generated given the options you selected. Customize the generated code to suit your needs.



Creating a DataBean

A DataBean is a Java class that models an observation in a SAS data set. It can be used to navigate record by record through a data set to view details of individual observations. For more information about DataBeans, see “Developing Data-Driven Applications Using JDBC and Java Servlet/JSP Technologies” available at <http://support.sas.com/rnd/appdev/doc/sugi28p49-28.pdf>.

The DataBean template in AppDev Studio 3.2 Eclipse Plug-ins supports only JDBC connections.

There are two ways you can create a DataBean:

- ☐ Create a SAS Java Project and then add content to it by selecting the **Add Template Content** check box at the end of the creation process.
- ☐ Create a new SAS Java Project and then add content to it later.

To create a DataBean, follow these steps:

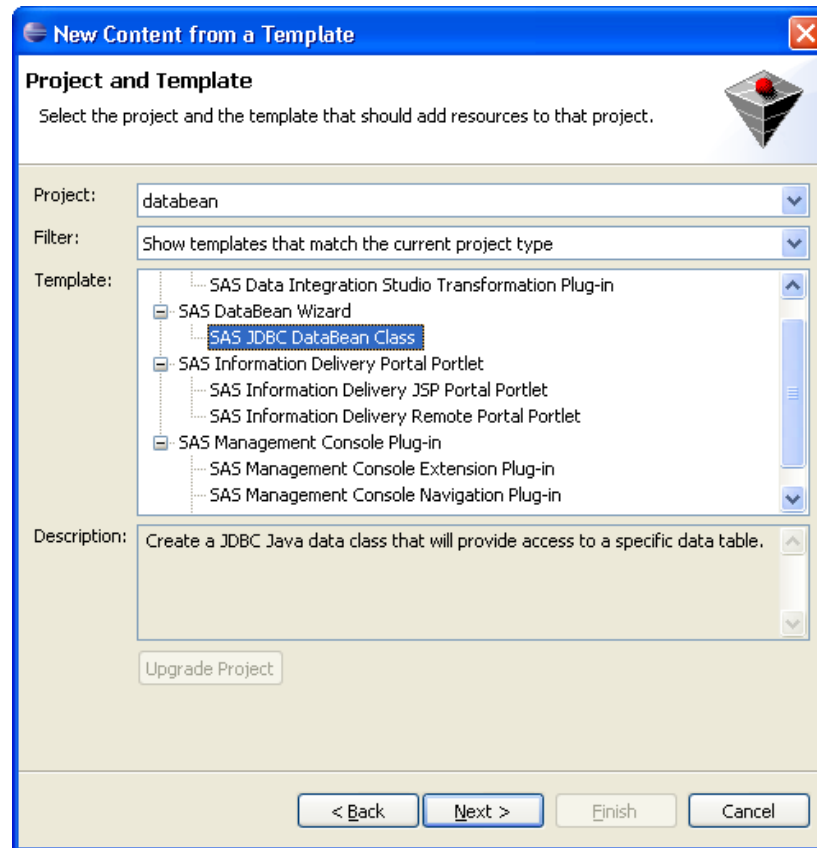
- 1 Specify separate source and output directories (see “Specifying Separate Java Source and Output Directories” on page 89).
- 2 Create a new SAS Java Project (see “Creating a SAS Java Project” on page 5).
- 3 Select **File ► New ► Other**.
- 4 Expand **SAS AppDev Studio**.
- 5 Select **Add Template Content to Project**.
- 6 Click **Next**.
- 7 In the **Project** field, select the SAS Java Project to which you want to add the template content (the new project created in step 2).
- 8 Expand **SAS DataBean Wizard**.

9 Select **SAS JDBC DataBean Class**.

The **Upgrade Project** button is enabled only if the project to which you are adding the DataBean is not a SAS Java Project. In such cases, clicking the **Upgrade Project** button adds to the project the SAS Jar Repository and the SAS Java Project builders.

CAUTION:

If you are adding a **SAS JDBC DataBean Class** template to a **Dynamic Web Project**, do not upgrade the project. Doing so results in an incomplete project. Instead, select any of the SAS Web Application templates and click the **Upgrade Project** button. After the project is upgraded, you can then add the SAS JDBC DataBean Class template. △



10 Click **Next**.

11 Modify the **File name prefix** to specify the base name for the generated classes.

12 Click **Next**.

13 Specify the package and connection information for the DataBean.

New Content from a Template

SAS DataBean Wizard -- JDBC Connection

Enter information below to define the JDBC connection. Fill in the username and password fields if these are required for your connection.

Additional Help >

Generated File Location Information

File name prefix: jdbc

Package: com.sas.databean

Driver Information

Select a driver: SAS IOM JDBC Driver

Driver class: com.sas.rio.MVADriver

Datasource URL: jdbc:sasiom://localhost:5310

User Information

Username:

Password:

< Back Next > Finish Cancel

14 Click **Next**.

15 Enter an SQL query to access data from the data source and then click **Submit Query**.

New Content from a Template

DataBean Wizard -- JDBC Column Selection

Submit a query to select database columns.

Additional Help >

Query: select * from sashelp.class

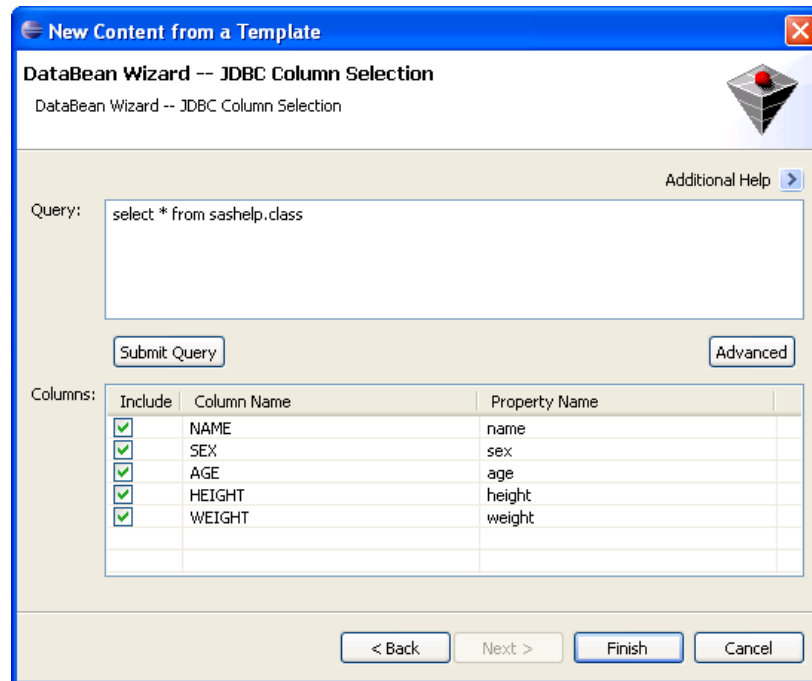
Submit Query Advanced

Columns:

Include	Column Name	Property Name

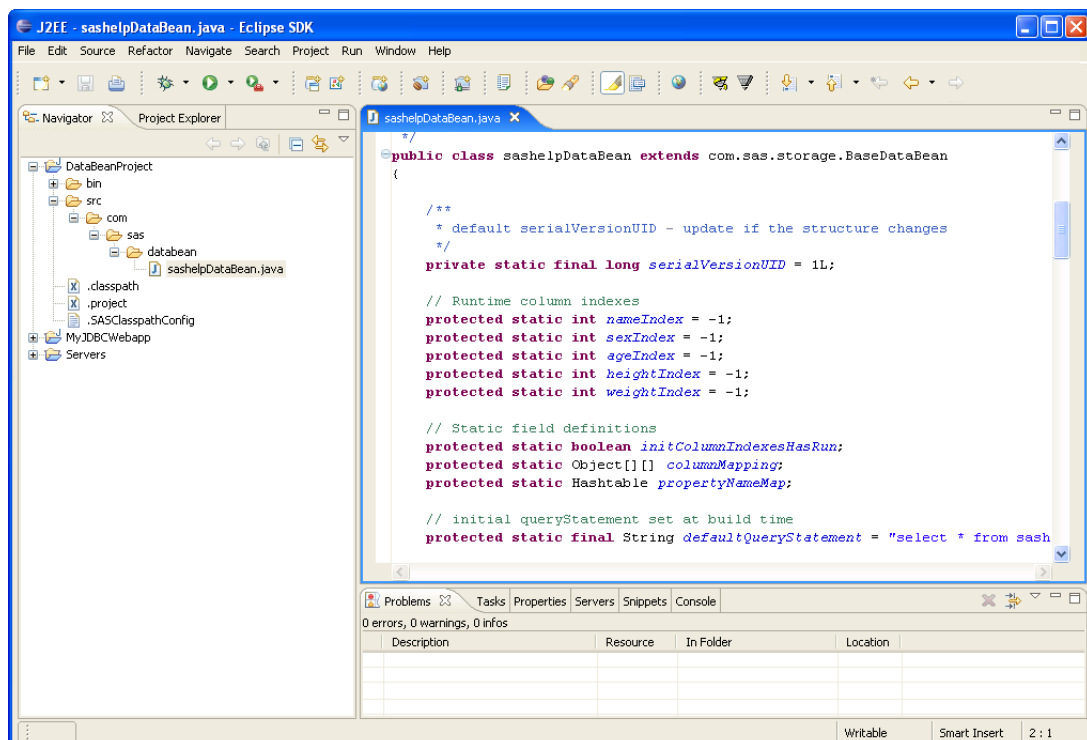
< Back Next > Finish Cancel

- 16 The columns from the data source are displayed in the **Columns** table. Specify the columns that you want to include in the DataBean and, for each column, provide the **Property Name** to which the column data should be mapped.



- 17 Click **Finish**.

The code to access the database table is generated given the options you selected. Customize the generated code to suit your needs.



Creating a JDBC Servlet

There are two JDBC Servlet templates:

- JDBC Default Servlet
- JDBC Tableview Servlet

Both of these templates can be added to a SAS Web Application Project or a Web Tools Dynamic Web Project.

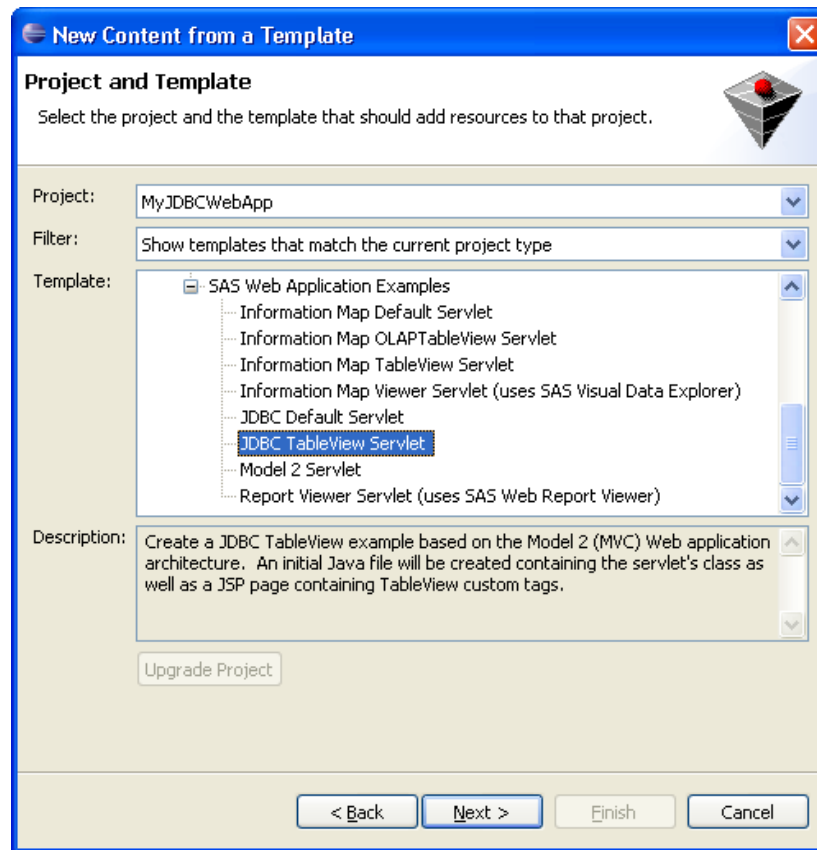
The following steps are common to creating a JDBC Servlet from a new project, but to run the servlet you must update at least one string in the generated code, and ensure that the server can run the project. These steps assume you have already configured a JDK and a Tomcat 4.1.18 Web server.

- 1 Specify separate source and output directories (see “Specifying Separate Java Source and Output Directories” on page 89).
- 2 Create a new SAS Web Application Project or a Web Tools Dynamic Web Project.
- 3 Select **File ► New ► Other**.
- 4 Expand **SAS AppDev Studio**.
- 5 Select **Add Template Content to Project**.
- 6 Click **Next**.
- 7 In the **Project** field, select the project to which you want to add the template content (the new project created in step 2).
- 8 If the project is a Web Tools Dynamic Web Project, the list of templates will be empty.

Change the **Filter** selection to **Show templates that match or can upgrade the current project** and the templates will appear.

- 9 Expand **SAS Web Application Examples**.

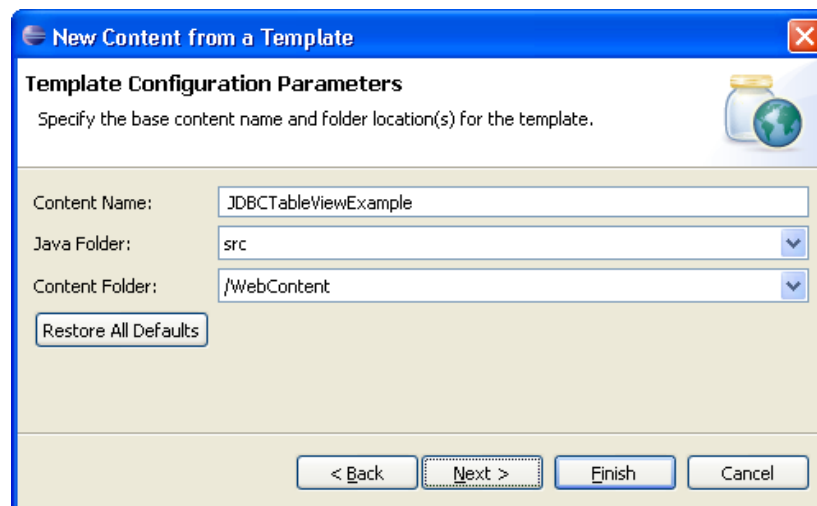
- 10 Select either the **JDBC Default Servlet** or the **JDBC TableView Servlet** template.



- 11 The **Upgrade Project** button is enabled if the project to which you are adding the JDBC servlet template is not a SAS Web Application Project but could be converted to one. Click the **Upgrade Project** button to convert a Web Tools Dynamic Web Project to a SAS Web Application Project.

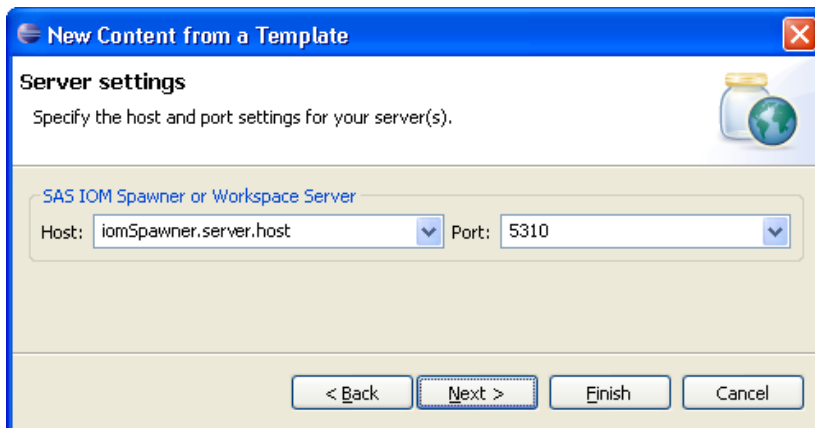
- 12 Click **Next**

- 13 The **Content Name** field specifies the base name of the generated classes. By changing this name, you can add multiple JDBC templates to the same project.



14 Click **Next**.

15 Specify the **Host** and **Port** of the SAS IOM Spawner or Workspace Server that you want the JDBC controller servlet to use.

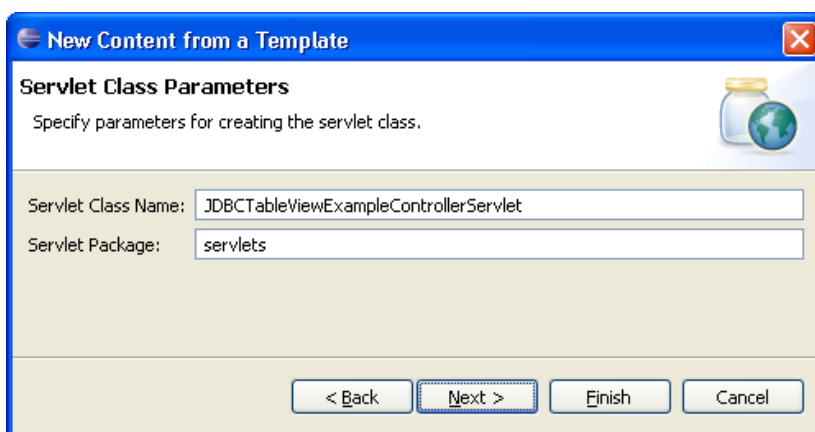


The screenshot shows a dialog box titled "New Content from a Template" with a close button (X) in the top right corner. The main heading is "Server settings" with a sub-instruction: "Specify the host and port settings for your server(s)." There is a small icon of a jar and a globe to the right. Below this, a section titled "SAS IOM Spawner or Workspace Server" contains two input fields: "Host:" with a dropdown menu showing "iomSpawner.server.host" and "Port:" with a dropdown menu showing "5310". At the bottom, there are four buttons: "< Back", "Next >" (highlighted with a dashed border), "Finish", and "Cancel".

16 Click **Next**.

If you do not need to configure the servlet, click **Finish**.

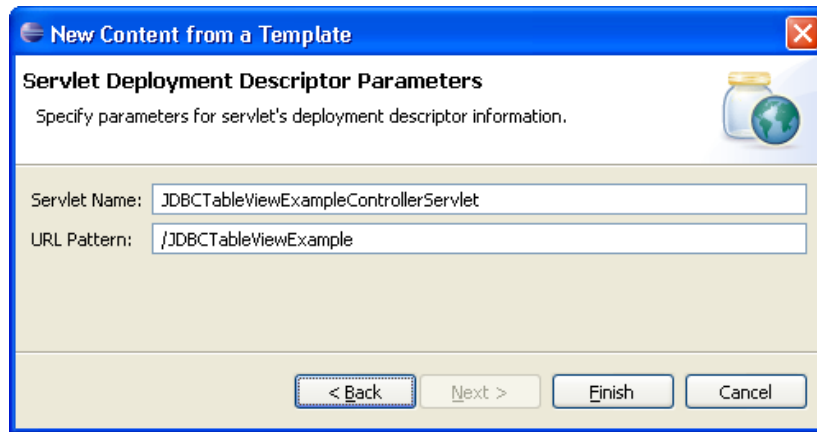
17 Specify the servlet class parameters that affect the creation of the JDBC controller servlet.



The screenshot shows the same dialog box, but the "Servlet Class Parameters" tab is selected. The sub-instruction is "Specify parameters for creating the servlet class." with the same jar and globe icon. Below, there are two text input fields: "Servlet Class Name:" containing "JDBCTableViewExampleControllerServlet" and "Servlet Package:" containing "servlets". At the bottom, the same four buttons are present, with "Next >" highlighted.

18 Click **Next**.

19 Specify the servlet deployment descriptor parameters that affect the deployment descriptor information added to the project's **web.xml** file.



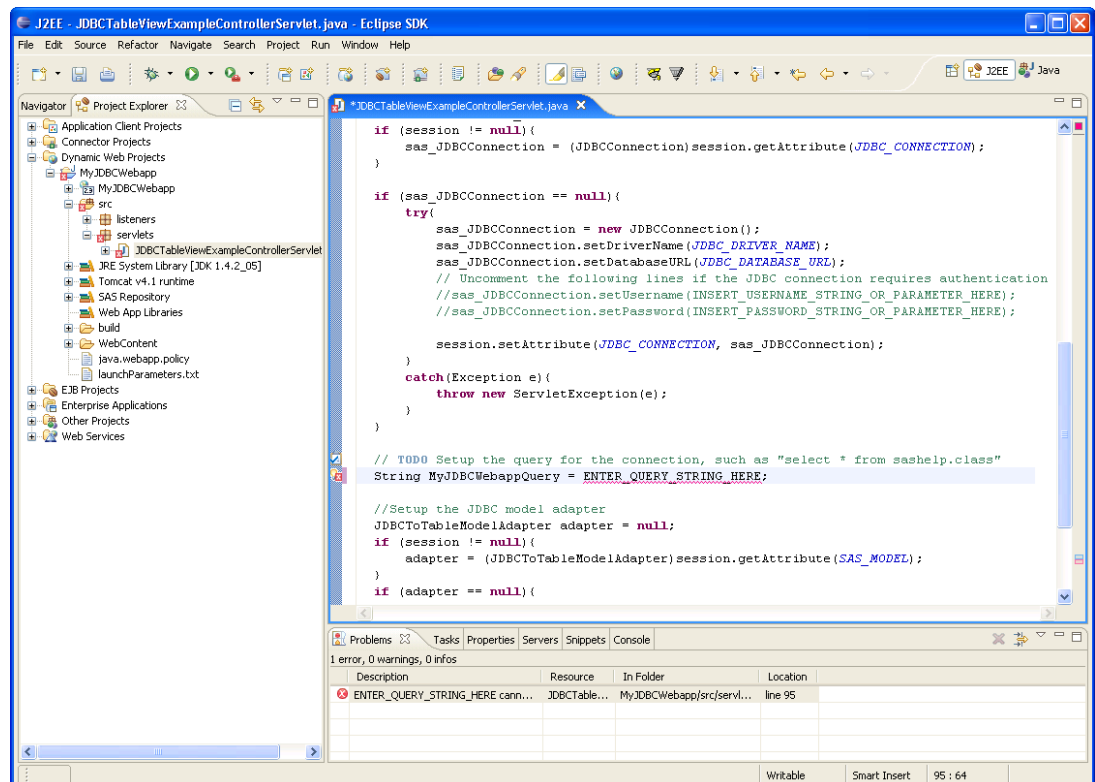
20 Click **Finish**.

The JDBC template content is generated given the options you specified, and is added to the project.

However, to run the template you must update at least one string in the generated code, and ensure that the server can run the project. Note the error in Display 3.1 on page 36 showing where a modification must be made to complete and run the template.

The preceding steps were common to both JDBC Servlet templates. For the steps specific to completing each template, see “Completing a JDBC Servlet” on page 37.

Display 3.1 JDBC Template Output with Error



Completing a JDBC Servlet

To run the code generated by either of the JDBC Servlet templates, you must update at least one string in the `JDBCTableViewExampleControllerServlet` file.

Regardless of the JDBC Servlet template you used, follow these steps:

- 1 Display the Problems view if it is not visible (select **Window ► Show View ► Problems**).
- 2 If an error with a description of `ENTER_QUERY_STRING_HERE` is not visible, rebuild the project by following these steps:
 - a Select from the Eclipse main menu **Project ► Clean**.
 - b Select **Clean projects selected below**.
 - c Select the check box for this project.
 - d Click **OK**.

Consider enabling automatic builds by selecting **Project ► Build Automatically**.

- 3 In the Problems view, double-click the error `ENTER_QUERY_STRING_HERE` to go to the error in the Java file.
- 4 Replace the `ENTER_QUERY_STRING_HERE` string with your query string. You can copy and paste the query string provided in the comment above the error, or enter your own query string.
- 5 Save the file.

If you used a JDBC Default Servlet, also follow these steps:

- 1 In the same servlet code (`JDBCTableViewExampleControllerServlet`), find the comment `//INSERT_MODEL_ADAPTER_CODE_HERE`.
- 2 Replace this comment with code to create a `JDBCAdapter` object to serve as the model for the viewer you plan to use.
- 3 Save the file.
- 4 Switch to the editor for the JSP created for the example. Where indicated, add the viewer tag(s) needed to display the model adapter created in the step above.
- 5 Save the file.

Preparing the Server to Run the JDBC Servlet

To run the JDBC Servlet application, ensure that the following are true:

- ☐ Permissions for the application have been added to the server's `catalina.policy` file. See “Adding Security Policies to the Web Application Server Policy File” on page 15.
- ☐ Parameters for the template have been added to the server's launch configuration, according to “Configuring the Web Application Server Launch Settings” on page 16.

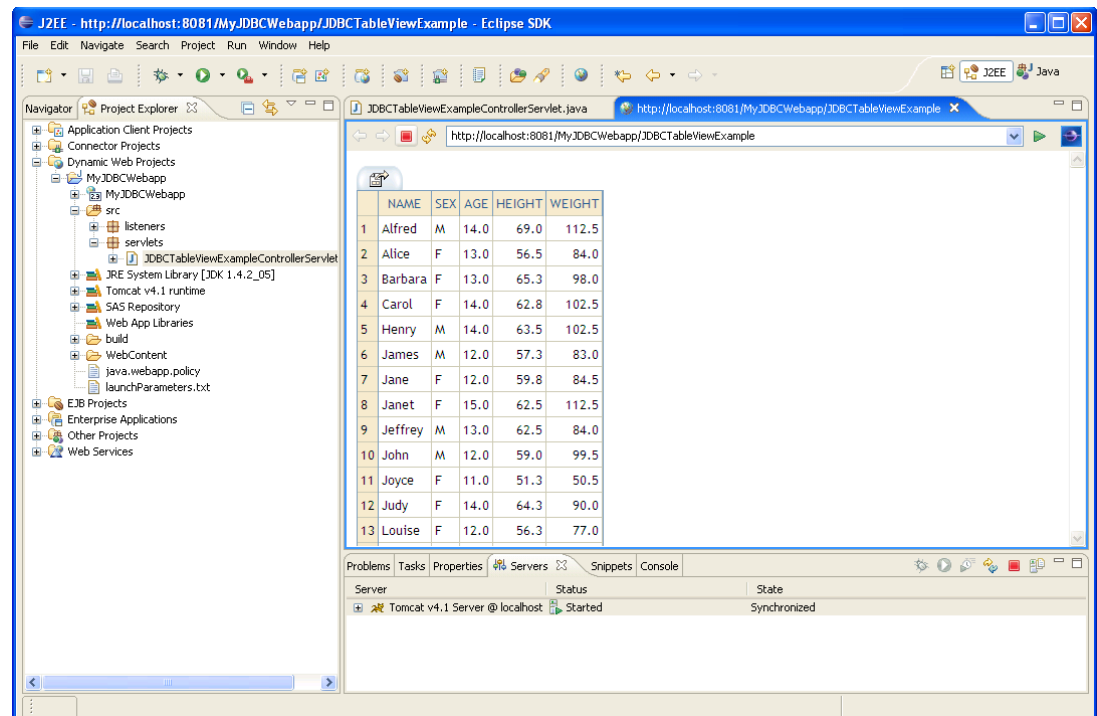
Running the JDBC Servlet

To run a JDBC Servlet application, follow these steps:

- 1 Start the IOM Spawner or SAS Workspace Server.
- 2 In the editor window for the controller servlet, right-click and select **Run As ► Run on Server**.
- 3 For host name select `localhost`. For the server, select **Tomcat v4.1 Server**.

4 Click **Finish**.

The Web application is published to the server and the server is started. After the server starts, a browser window displays the response from executing the controller servlet. The JDBC TableView Servlet template should resemble the following:



Using the Information Map Servlet Templates

There are four templates related to Information Maps:

- ☐ Information Map Default Servlet
- ☐ Information Map OLAPTableView Servlet
- ☐ Information Map TableView Servlet
- ☐ Information Map Viewer Servlet (uses SAS Visual Data Explorer)

All of these templates can be added to a SAS Web Application Project or a Web Tools Dynamic Web Project.

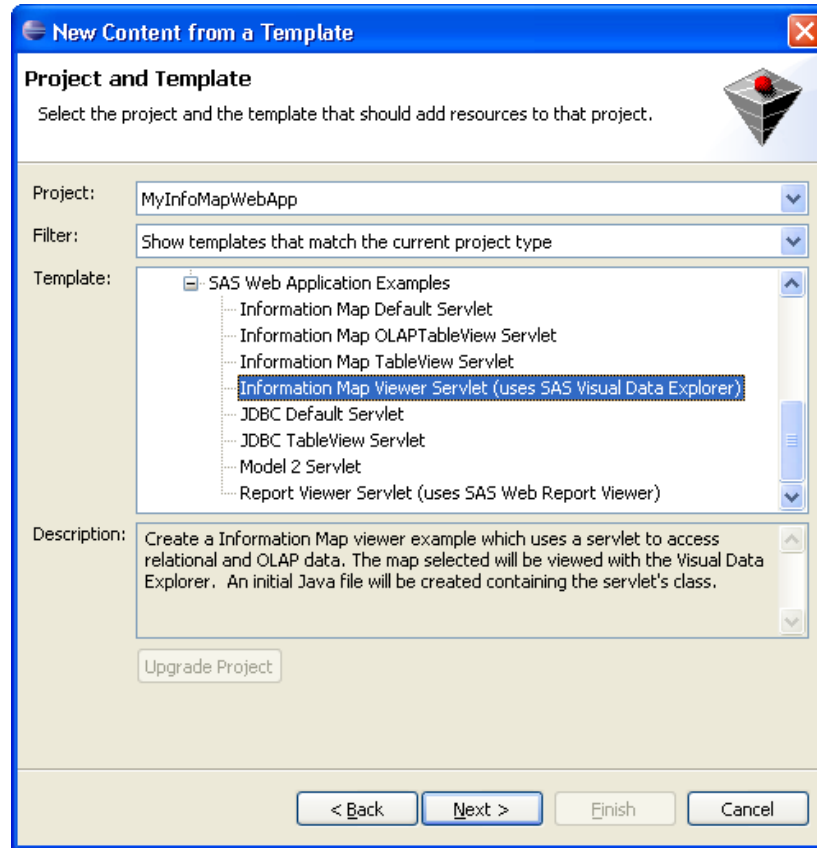
The following steps are common to adding any of the Information Map Servlet Templates to a project. Steps for finishing each particular type of Information Map Servlet Template are listed later. These steps assume you have already configured a JDK and a Tomcat 4.1.18 Web server.

- 1 Select the project to which you want to add the template by selecting the project or one of its resources in a navigator or editor.
- 2 Select **File ► New ► Other**.
- 3 Expand the **SAS AppDev Studio** folder.
- 4 Select **Add Template Content to Project**.
- 5 Click **Next**.
- 6 If the selected project is a Web Tools Dynamic Web Project, the list of templates will be empty. You must upgrade the project to a SAS Web Application Project

before a template can be added to it. If this is the case, change the Filter selection to **Show templates that match or can upgrade the current project** and templates will appear in the list.

- 7 Expand **SAS Java Web Application** and then **SAS Web Application Examples** in the tree, if not already expanded.

- 8 Select one of the Information Map Viewer Servlet templates.



- 9 If the project you are adding the template to is a Dynamic Web Project, an error is displayed indicating that the project does not currently meet the template's requirements. In addition, the **Upgrade Project** button is enabled. Click **Upgrade Project** to add the required resources to the project.
- 10 Click **Next**.
- 11 Modify the **Content Name** to specify the base name some of the generated classes.
By changing the content name, you can add multiple Information Map examples to the same Web project.

New Content from a Template

Template Configuration Parameters

Specify the base content name and folder location(s) for the template.

Content Name:

Java Folder:

Content Folder:

12 Click **Next**.

13 Specify the SAS Metadata Server settings. The Metadata Server Host defaults to your system's machine name.

If you previously entered these server settings while adding a different template to the project, those values are entered as defaults. Adjust them as needed.

After you type in a password, the **Test Configuration** button is enabled.

New Content from a Template

Foundation Services settings

Specify your Metadata Server settings and Foundation Services deployment(s). To enable browsing and testing, complete all metadata server fields.

Metadata Server

Host: Port:

User ID: Password:

Authentication Domain:

Repository Name:

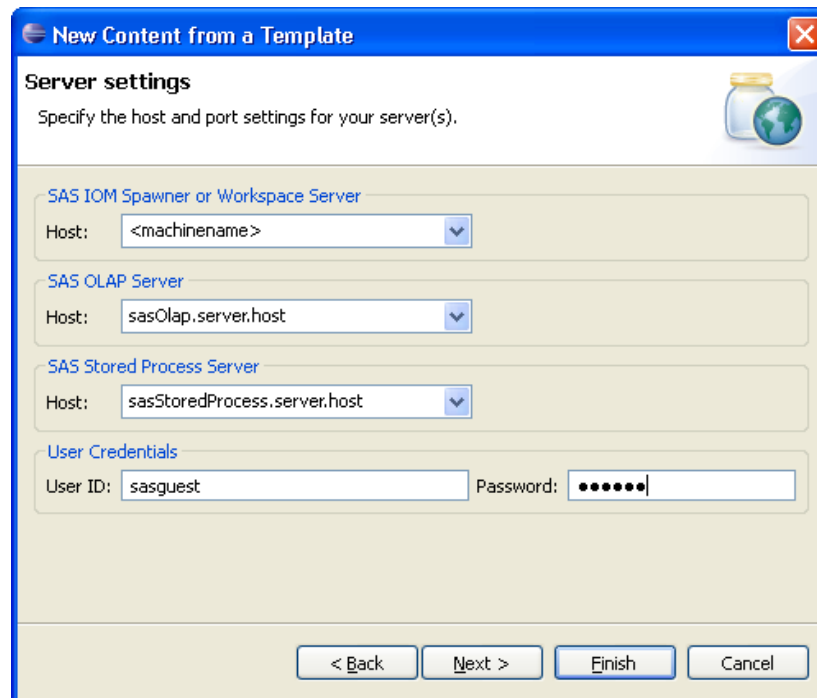
- 14 If the specified SAS Metadata Server is running, you can click the **Test Configuration** button to verify that the ADS Local Services service deployment is present on the server. You can specify a different service deployment by clicking on the **Advanced>>** button and specifying the desired service deployment.

For more information, see “Importing the ADS Local Services into a SAS Metadata Repository” on page 80.

- 15 Click **Next**.

- 16 Specify the hosts for the SAS IOM Spawner or Workspace, OLAP, and Stored Process servers. These all default to your system’s machine name. Also specify the User ID and Password to be used by the controller servlet.

If you previously entered these server settings while adding a different template to the project, those values are entered as defaults. Adjust them as needed.



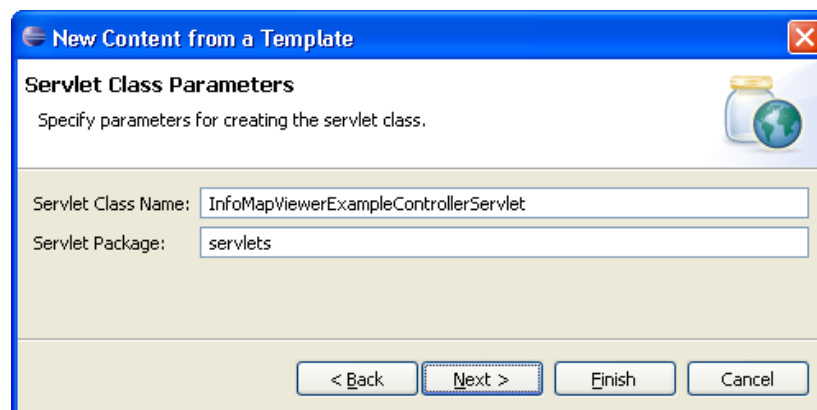
The screenshot shows the 'New Content from a Template' dialog box with the 'Server settings' tab selected. The dialog has a blue title bar and a close button in the top right. Below the title bar is a header area with the text 'Specify the host and port settings for your server(s)' and a small icon of a globe and a server. The main area contains four sections: 'SAS IOM Spawner or Workspace Server' with a 'Host' dropdown set to '<machinename>', 'SAS OLAP Server' with a 'Host' dropdown set to 'sasOlap.server.host', 'SAS Stored Process Server' with a 'Host' dropdown set to 'sasStoredProcess.server.host', and 'User Credentials' with 'User ID' set to 'sasguest' and 'Password' masked with dots. At the bottom are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

- 17 If you do not need to configure the controller servlet, click **Finish**.

Otherwise, click **Next**.

- 18 Specify the servlet customization information.

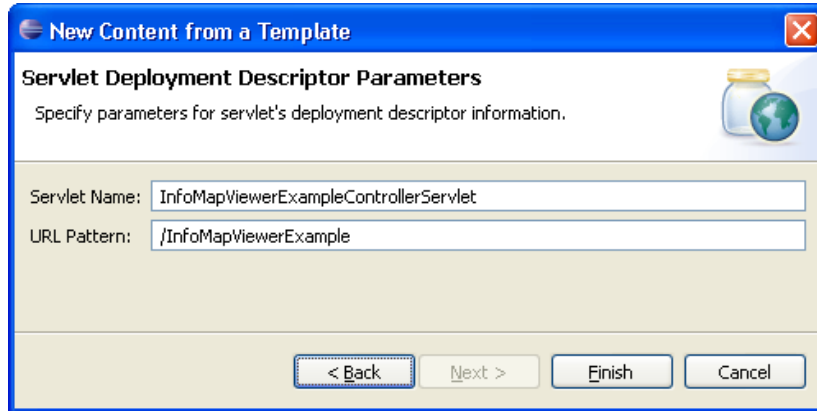
These settings affect the creation of the Information Map controller servlet.



The screenshot shows the 'New Content from a Template' dialog box with the 'Servlet Class Parameters' tab selected. The dialog has a blue title bar and a close button in the top right. Below the title bar is a header area with the text 'Specify parameters for creating the servlet class.' and a small icon of a globe and a server. The main area contains two sections: 'Servlet Class Name' with a text field containing 'InfoMapViewExampleControllerServlet' and 'Servlet Package' with a text field containing 'servlets'. At the bottom are four buttons: '< Back', 'Next >', 'Finish', and 'Cancel'.

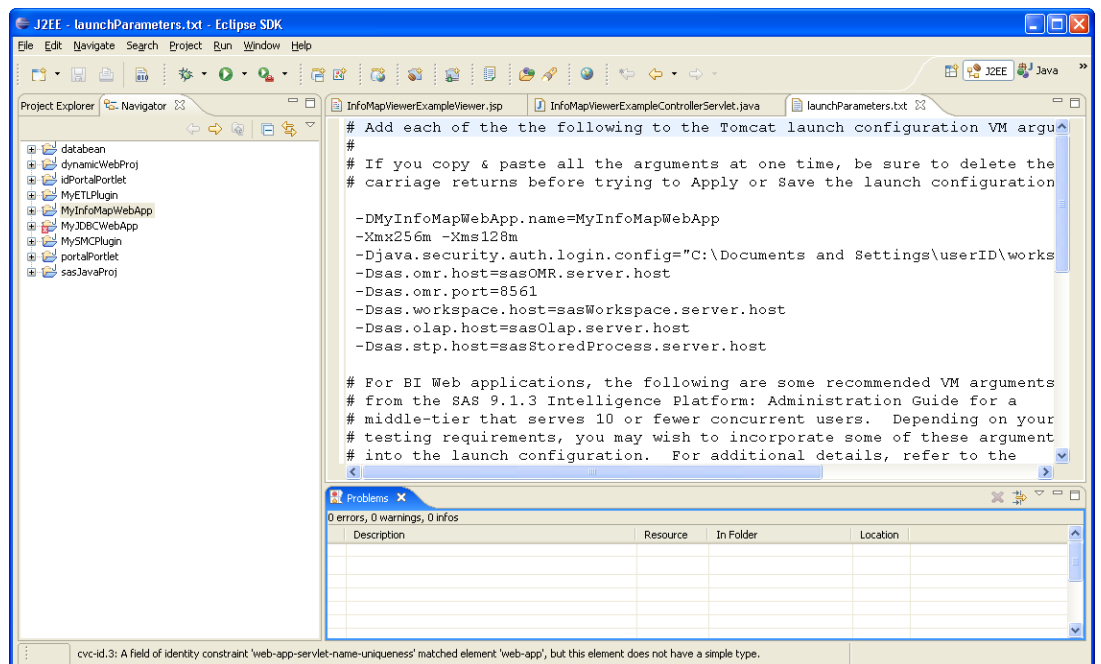
19 Click **Next**.

20 Specify the settings that affect how the deployment descriptor information is added to the Web project's **web.xml** file.



21 Click **Finish**.

The Information Map template content is added to the project.



Refer to the sections below for steps to complete the specific Information Map templates.

Completing an Information Map Viewer Servlet

To run the servlet, you might need to update a string in the controller servlet class. The static string **DEFAULT_REPOSITORY_FOLDER** defines the starting folder for the remote file selector displayed by the Information Map Viewer Servlet Template. The default value of this string is **SBIP://Foundation/BIP Tree(Folder)**, and should

exist in most installations. If you want to start in a different folder, or the default folder does not exist in the SAS Metadata Repository, you must edit the string value.

To edit the value of the **DEFAULT_REPOSITORY_FOLDER** string, follow these steps:

- 1 Open the **InfoMapViewExampleControllerServlet.java** file.
- 2 Locate the string **DEFAULT_REPOSITORY_FOLDER**.
- 3 Edit the value of the string.
- 4 Save the file.

Completing an Information Map TableView or OLAPTableView Servlet

To run either servlet, you must enter the location of the Relational Information Map or OLAP Information Map, depending on the type of template you added. Follow these steps:

- 1 Display the Problems view if it is not visible (select **Window ► Show View ► Problems**).
- 2 If an error with a description of **ENTER_MAP_NAME_HERE cannot be resolved** is not visible, rebuild the project by following these steps:
 - a Select from the Eclipse main menu **Project ► Clean**.
 - b Select **Clean projects selected below**.
 - c Select the check box for this project.
 - d Click **OK**.

Consider enabling automatic builds by selecting **Project ► Build Automatically**.

- 3 In the Problems view, double-click the error **ENTER_MAP_NAME_HERE cannot be resolved** to go to the error in the Java file.
- 4 Replace the **ENTER_MAP_NAME_HERE** text with the path to the Relational Information Map or OLAP Information Map you want to display.
- 5 Save the file.

Completing an Information Map Default Servlet

The Information Map Default Servlet template provides a minimal implementation patterned after the other Information Map templates. To complete and run the servlet, follow these steps:

- 1 In the servlet code, find the comment **//INSERT_MODEL_ADAPTER_CODE_HERE**.
- 2 Replace this comment with code to create a **BusinessQueryAdapter** object to serve as the model for the viewer you plan to use.
- 3 Save the file.
- 4 Switch to the editor for the JSP created for the example. Where indicated, add the viewer tag(s) needed to display the model adapter created above.
- 5 Save the file.

Preparing the Server to Run the Information Map Servlet Application

To run the Information Map Servlet application, ensure that the following are true:

- ☐ Permissions for the application have been added to the server's catalina.policy file. See "Adding Security Policies to the Web Application Server Policy File" on page 15.

- Parameters for the template have been added to the server's launch configuration, according to "Configuring the Web Application Server Launch Settings" on page 16.

Running an Information Map Servlet Web Application

To run an Information Map Servlet application, follow these steps:

- 1 Start the SAS Metadata Server and IOM Spawner or SAS Workspace Server. If the servlet will display an OLAP table, also start the OLAP server.
- 2 In the editor window for the controller servlet, right-click and select **Run As ► Run on Server**.
- 3 For host name select **localhost**. For the server, select **Tomcat v4.1 Server**.
- 4 Click **Finish**.

The Web application is published to the server and the server is started. After the server starts, a browser window displays the response from executing the controller servlet.

Using the Report Viewer Servlet Template

There is only one Report Viewer template: Report Viewer Servlet (uses SAS Web Report Viewer). This template can be added to a SAS Web Application Project or a Web Tools Dynamic Web Project. These steps assume you have already configured a JDK and a Tomcat 4.1.18 Web server.

To use the Report Viewer Servlet template, the SAS Web Application Infrastructure Kit and the SAS Web Report Viewer must be installed.

There are no restrictions on where the SAS Web Application Infrastructure Kit, SAS Web Report Viewer, and AppDev Studio 3.2 Eclipse Plug-ins are installed. They can all be on different systems, the same system, or a combination of systems. The only requirement is that they all must use the same SAS Metadata Server and Remote Services deployment.

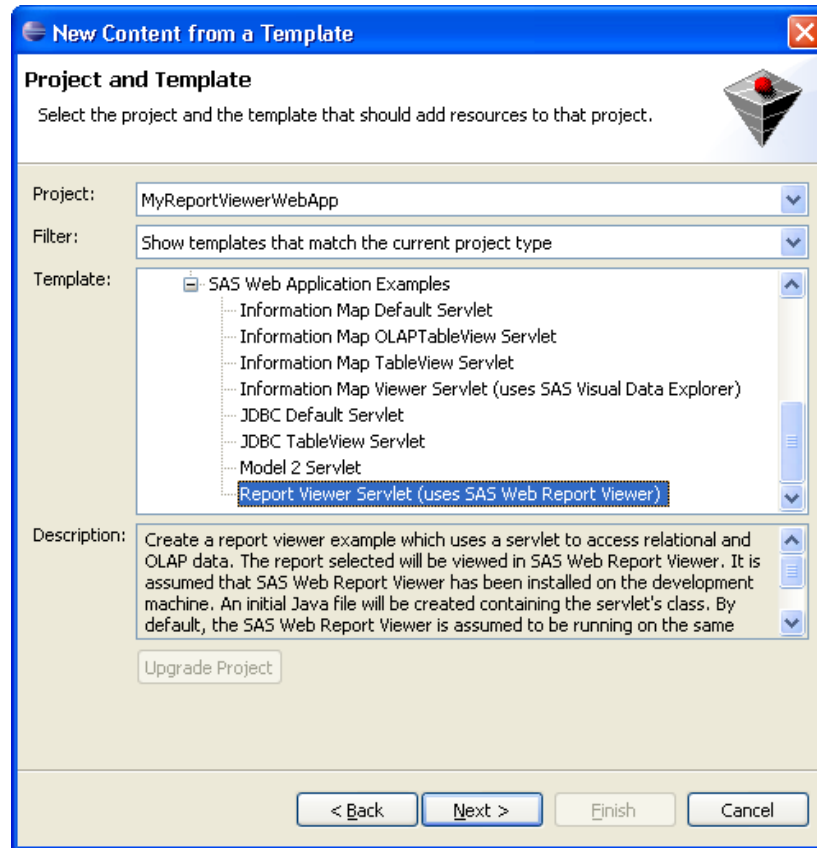
In addition, the SAS Metadata Repository must contain the Remote Services and the ADS Local Services. Importing the Remote Services should have been done as part of the post-installation for the SAS Web Application Infrastructure Kit.

The servers specified in the template must match those that are configured in SAS Metadata Repository. For example, the DAV Server host must be same host as the DAV Server in the repository, and the SAS Remote Services Server host must be the same host used in the Remote Services.

Follow these steps to add a Report Viewer template to a project:

- 1 Select the project to which you want to add the template by selecting the project or one of its resources in a navigator or editor.
- 2 Select **File ► New ► Other**.
- 3 Expand the **SAS AppDev Studio** folder.
- 4 Select **Add Template Content to Project**.
- 5 Click **Next**.
- 6 If the selected project is a Web Tools Dynamic Web Project, the list of templates will be empty. You must upgrade the project to a SAS Web Application Project before a template can be added to it. If this is the case, change the Filter selection to **Show templates that match or can upgrade the current project** and templates will appear in the list.
- 7 Expand **SAS Java Web Application** and then **SAS Web Application Examples** in the tree, if not already expanded.

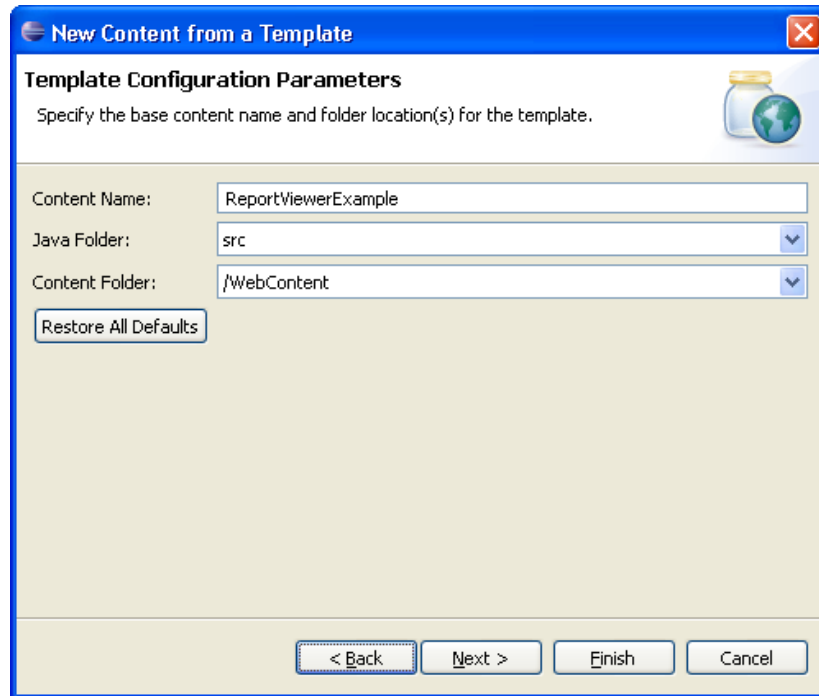
- 8 Select the **Report Viewer Servlet** template.



- 9 If the project you are adding the template to is a Dynamic Web Project, an error is displayed indicating that the project does not currently meet the template's requirements. In addition, the **Upgrade Project** button is enabled. Click **Upgrade Project** to add the required resources to the project.
- 10 Click **Next**.

- 11 Modify the **Content Name** to specify the base name some of the generated classes.

By changing the content name, you can add multiple Report Viewer templates to the same Web project.



New Content from a Template

Template Configuration Parameters
Specify the base content name and folder location(s) for the template.

Content Name: ReportViewerExample

Java Folder: src

Content Folder: /WebContent

Restore All Defaults

< Back Next > Finish Cancel

- 12 Click **Next**.

- 13 Specify the SAS Metadata Server settings. The Metadata Server Host defaults to your system's machine name.

If you previously entered these server settings while adding a different template to the project, those values are entered as defaults. Adjust them as needed.

After you type in a password, the **Test Configuration** button is enabled.

- 14** If the specified SAS Metadata Server is running, you can click the **Test Configuration** button to verify that the ADS Local Services service deployment is present on the server. You can specify a different service deployment by clicking on the **Advanced>>** button and specifying the desired service deployment.

See also “Importing the ADS Local Services into a SAS Metadata Repository” on page 80.

It is assumed that the Remote Services were imported into the SAS Metadata Server by some other SAS installation.

- 15** Click **Next**.

- 16** Specify the hosts for the Workspace, OLAP, and Stored Process servers. These all default to your system’s machine name. Also specify the User ID and Password to be used by the controller servlet.

If you previously entered these server settings while adding a different template to the project, those values are entered as defaults. Adjust them as needed.

New Content from a Template

Server settings
Specify the host and port settings for your server(s).

SAS IOM Spawner or Workspace Server
Host: <machinename>

SAS OLAP Server
Host: sasOlap.server.host

SAS Stored Process Server
Host: sasStoredProcess.server.host

SAS Remote Services Server
Host: sasRemote.server.host

DAV Server
Host: dav.server.host Port: 80

SAS Web Report Viewer Server
Host: sasWebRptV.server.host Port: 3080

User Credentials
User ID: sasguest Password: •••••••

< Back Next > Finish Cancel

17 If you do not need to configure the controller servlet, click **Finish**.

Otherwise, click **Next**.

18 Specify the servlet customization information.

These settings affect the creation of the Report Viewer controller servlet.

New Content from a Template

Servlet Class Parameters
Specify parameters for creating the servlet class.

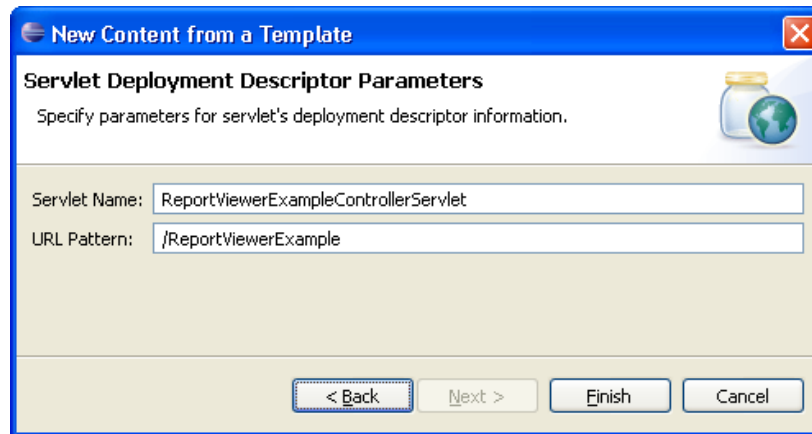
Servlet Class Name: ReportViewerExampleControllerServlet

Servlet Package: servlets

< Back Next > Finish Cancel

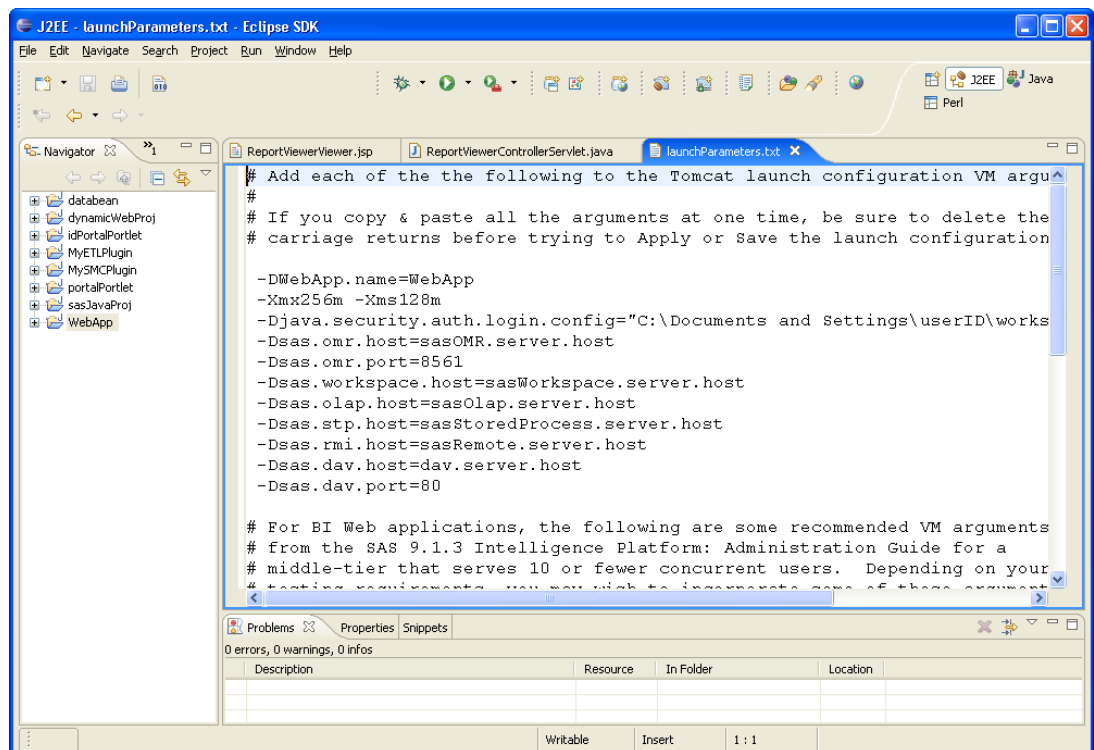
19 Click **Next**.

- 20 Specify the settings that affect how the deployment descriptor information is added to the Web project's **web.xml** file.



- 21 Click **Finish**.

The Report Viewer template content is added to the project.



Refer to the sections below for steps to complete the Report Viewer template.

Completing a Report Viewer Servlet

To run the servlet, you might have to update a string in the controller servlet class (ReportViewerExampleControllerServlet). The static string **DEFAULT_REPOSITORY_FOLDER** defines the starting folder for the remote file selector

displayed by the Report Viewer Servlet template. The default value for this string is **SBIP://Foundation/BIP Tree(Folder)**, and should exist in most installations. If you want to start in a different folder, or if the default folder does not exist in the SAS Metadata Repository, you must edit the string value.

To edit the value of the **DEFAULT_REPOSITORY_FOLDER** string, follow these steps:

- 1 Open the **ReportViewerExampleControllerServlet.java** file.
- 2 Locate the string **DEFAULT_REPOSITORY_FOLDER**.
- 3 Edit the value of the string.
- 4 Save the file.

Preparing the Server to Run the Report Viewer Application

To run the Report Viewer Servlet application, ensure that the following are true:

- ☐ Permissions for the application have been added to the server's **catalina.policy** file. See "Adding Security Policies to the Web Application Server Policy File" on page 15.
- ☐ Parameters for the template have been added to the server's launch configuration, according to "Configuring the Web Application Server Launch Settings" on page 16.

Running the Report Viewer Application

To run a Report Viewer application, follow these steps:

- 1 Start the SAS Metadata Server, IOM Spawner or SAS Workspace Server, SAS Remote Services Application, and the web server running the Web Report Viewer Web application. If the servlet will display an OLAP table, also start the OLAP server.
- 2 In the editor window for the controller servlet, right-click and select **Run As ► Run on Server**.
- 3 For host name select **localhost**. For the server, select **Tomcat v4.1 Server**.
Select the **Set server as project default** check box to avoid making this choice each time you run the application.
- 4 Click **Finish**.

The Web application is published to the server and the server is started. After the server starts, a browser window displays the response from executing the controller servlet.

Using the Context Listener Templates

There are three templates related to Context Listeners:

- ☐ Context Listener for Default Services
- ☐ Context Listener for Local Services
- ☐ Context Listener for Remote & Local Services

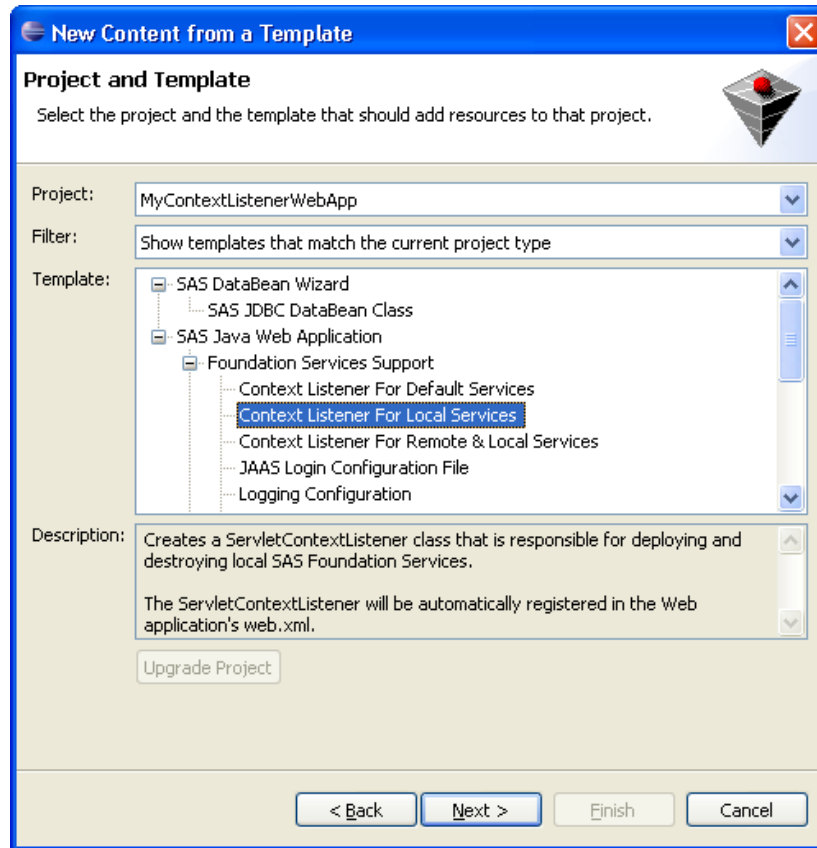
All of these templates can be added to a SAS Web Application Project or a Web Tools Dynamic Web Project.

The Context Listeners work to deploy and destroy local and remote Foundation Services. For information on how the templates differ, see "Foundation Services Context Listener Class" on page 84.

All the Context Listener templates are added using the following procedure:

- 1 Select the project to which you want to add the template by selecting the project or one of its resources in a navigator or editor.
- 2 Select **File ► New ► Other**.
- 3 Expand the **SAS AppDev Studio** folder.
- 4 Select **Add Template Content to Project**.
- 5 Click **Next**.
- 6 If the selected project is a Web Tools Dynamic Web Project, the list of templates will be empty. You must upgrade the project to a SAS Web Application Project before a template can be added to it. If this is the case, change the Filter selection to **Show templates that match or can upgrade the current project** and templates will appear in the list.
- 7 Expand **SAS Java Web Application** and then **Foundation Services Support** in the tree, if not already expanded.

- 8 Select one of the Context Listener templates.



- 9 If the project you are adding the template to is a Dynamic Web Project, an error is displayed indicating that the project does not currently meet the template's requirements. In addition, the **Upgrade Project** button is enabled. Click **Upgrade Project** to add the required resources to the project.
- 10 Click **Next**.

- 11 Modify the **Context listener class file** and **Java Package** fields, if necessary. If you are adding a *Context Listener For Default Services* template, the **Content Folder** field will not be available because that template has no content to add.

New Content from a Template

Template Configuration Parameters
Create a Foundation Services ServletContextListener to deploy local services.

Context listener class file: FoundationServicesContextListener.java

Java Folder: src

Java Package: listeners

Content Folder: /WebContent

Restore All Defaults

< Back Next > Finish Cancel

- 12 If you selected the *Context Listener For Default Services* template, click **Finish**. Otherwise, click **Next**.
- 13 Specify the SAS Metadata Server settings. The Metadata Server Host defaults to your system's machine name.
- If you previously entered these server settings while adding a different template to the project, those values are entered as defaults. Adjust them as needed.
- After you type in a password, the **Test Configuration** button is enabled.

New Content from a Template

Foundation Services settings

Specify your Metadata Server settings and Foundation Services deployment(s). To enable browsing and testing, complete all metadata server fields.

Metadata Server

Host: <machinename> Port: 8561

User ID: saswbadm Password: ●●●●●●

Authentication Domain: DefaultAuth

Repository Name: Foundation Browse...

Advanced >>

Test Configuration...

< Back Next > Finish Cancel

- 14 If the specified SAS Metadata Server is running, you can click the **Test Configuration** button to verify that the ADS Local Services service deployment is present on the server. You can specify a different service deployment by clicking on the **Advanced>>** button and specifying the desired service deployment.

If you are adding a *Context Listener for Remote & Local Services* template, the Remote Services service deployment will also be verified.

For more information, see “Importing the ADS Local Services into a SAS Metadata Repository” on page 80.

- 15 If you are adding a Context Listener For Local Services, click **Finish**.

Otherwise, click **Next**.

- 16 Specify the host for the SAS Remote Services Server.

New Content from a Template

Server settings

Specify the host and port settings for your server(s).

SAS Remote Services Server

Host: <machinename>

< Back Next > Finish Cancel

- 17 Click **Finish**.

A listener is used by a SAS Web Application template (all Information Map, JDBC, and Report View templates) only if the listener meets the required functional level for

that SAS Web Application template (see “Foundation Services Context Listener Class” on page 84).

If the listener does not meet the required functional level for a SAS Web Application template, the listener is upgraded (the Java file you created with the Context Listener template is upgraded). A listener can be upgraded only to the next functional level. Downgrading is not supported.

However, you can add a Context Listener template to a SAS Web Application Project multiple times to create a new listener without upgrading it.

If you change the Context listener class file, the Java Package, or the SAS Metadata server or SAS Remote Services settings when adding an additional Content Listener template to a project, a new listener is created using the new values, and the existing listener file is discarded (but not deleted). The project’s Web application is modified to use the new listener file and the old one is not used.

Regardless of how many times the listener file is altered or upgraded, there is only one element in the project’s **web.xml** file that specifies a **ServletContextListener** that deploys and destroys SAS Foundation Services.

If you modify the defaults for the SAS Metadata server and/or services deployments, those new values become project-wide defaults. Any template that queries for these SAS Metadata Server settings will display the new values as the defaults.

Using the Logging Configuration Template

There is only one Logging Configuration template. This template can be added to a SAS Web Application or a Web Tools Dynamic Web Project.

The Logging Configuration template is useful for controlling the logging configuration from within a Web application, rather than by modifying the logging configuration stored in the services metadata that the Web application deploys.

Additional information about using the logging configuration file is included in comments within the file (**LoggingServiceConfig.xml**). As mentioned there, information about the format of this configuration file can be found in the API documentation for the **com.sas.services.logging** package.

To access this package, follow these steps:

- 1 Select **Help ► Help Contents**.
- 2 Click **SAS Components Guide**.
- 3 Expand **Reference** and then **Foundation API Reference**.
- 4 Click **com.sas.services.logging**.

This template requires a Context Listener in the SAS Web application. The context listener finds the Logging Configuration configuration file using a context parameter specified in the **web.xml** file, which is added or updated by the Logging Configuration template.

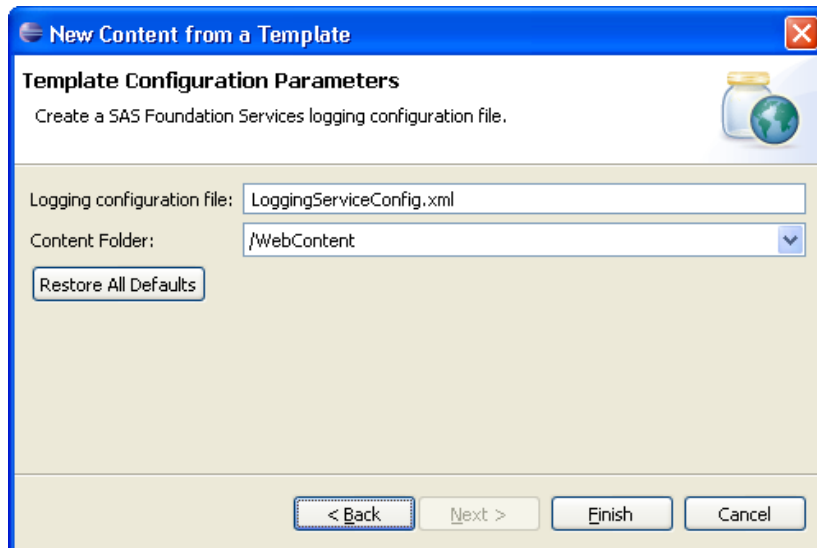
You can use the Logging Configuration template to add multiple logging configuration files to your SAS Web application. The template will never overwrite a logging configuration file, but it will update the **web.xml** file to use the file specified. Thus, you can switch between multiple logging configuration files by using this template and specifying the name of the existing configuration file you want to use.

To add a Logging Configuration template to a project, follow these steps:

- 1 Select the project to which you want to add the template by selecting the project or one of its resources in a navigator or editor.
- 2 Select **File ► New ► Other**.
- 3 Expand the **SAS AppDev Studio** folder.

- 4 Select **Add Template Content to Project**.
- 5 Click **Next**.
- 6 If the selected project is a Web Tools Dynamic Web Project, the list of templates will be empty. You must upgrade the project to a SAS Web Application Project before a template can be added to it. If this is the case, change the Filter selection to **Show templates that match or can upgrade the current project** and templates will appear in the list.
- 7 Expand **SAS Java Web Application** and then **Foundation Services Support** in the tree, if they are not already expanded.
- 8 Select the **Logging Configuration** template.
- 9 If the project you are adding the template to is a Dynamic Web Project, an error is displayed indicating that the project does not currently meet the template's requirements. In addition, the **Upgrade Project** button is enabled. Click the **Upgrade Project** button to add the required resources to the project.
- 10 Click **Next**.
- 11 Specify the name of the Logging configuration file.

The Logging configuration file will be placed in the **WEB-INF** folder under the selected Content Folder.



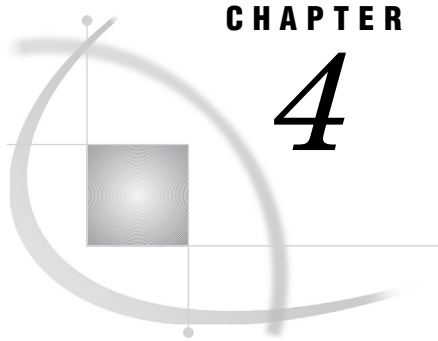
New Content from a Template

Template Configuration Parameters
Create a SAS Foundation Services logging configuration file.

Logging configuration file:

Content Folder:

- 12 Click **Finish**.



CHAPTER

4

Using the SAS Editor Extensions

<i>Introduction to SAS Editor Extensions</i>	59
<i>Accessing SAS Component API Documentation</i>	59
<i>Eclipse Help System</i>	60
<i>Context-Sensitive Javadoc in an Eclipse Window (F1)</i>	60
<i>Context-Sensitive Javadoc in an External Browser (Shift+F2)</i>	60
<i>Configuring External Javadoc for use within Eclipse</i>	60
<i>Adding Missing Import Statements</i>	60
<i>Using a SAS Import from Repository Quick Fix</i>	60
<i>Using the Organize SAS Imports Action</i>	61
<i>Attaching a SAS Model to a Viewer</i>	61
<i>Overview of Model/Viewer Connections</i>	61
<i>Enabling the Quick Assist Icons</i>	62
<i>Supported Viewers for Model/Viewer Connection Quick Assist</i>	62
<i>Using the SAS Model/Viewer Connection Quick Assist</i>	62
<i>SAS Snippets</i>	64
<i>Enabling SAS Snippets</i>	64
<i>Using SAS Snippets</i>	64

Introduction to SAS Editor Extensions

The SAS AppDev Studio 3.2 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

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 Click **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 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)

To view the Javadoc for a class in an external browser, place the insertion point on a class name in the source code and press 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 Specify the location of the Javadoc.

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

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, you should use the SAS AppDev Studio Eclipse Plug-ins Repository Quick Fix and the Organize SAS Imports action because both are aware of the SAS Versioned Jar Repository.

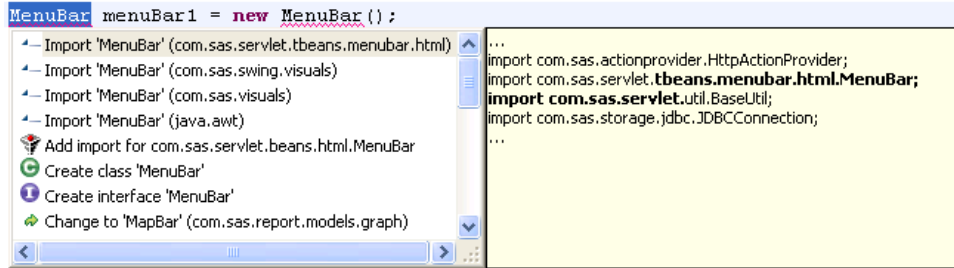
Using a SAS Import from Repository Quick Fix

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 decide to 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 desired Quick Fix and press ENTER. If you use a SAS Import from Repository Quick Fix, the entire project is rebuilt because the classpath has changed (if the auto-build option is enabled).

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 find a class exists in one of two states:

- ❑ the class is currently on the project classpath, but it has not been imported in 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

The SAS AppDev Studio 3.2 Eclipse Plug-ins do not support the ability to attach models to viewers using a drag-and-drop operation. However, the 3.2 release does provide an editor Quick Assist to facilitate 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`

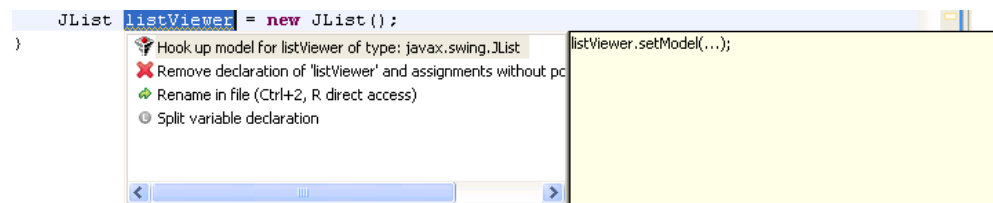
Using the SAS Model/Viewer Connection Quick Assist

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

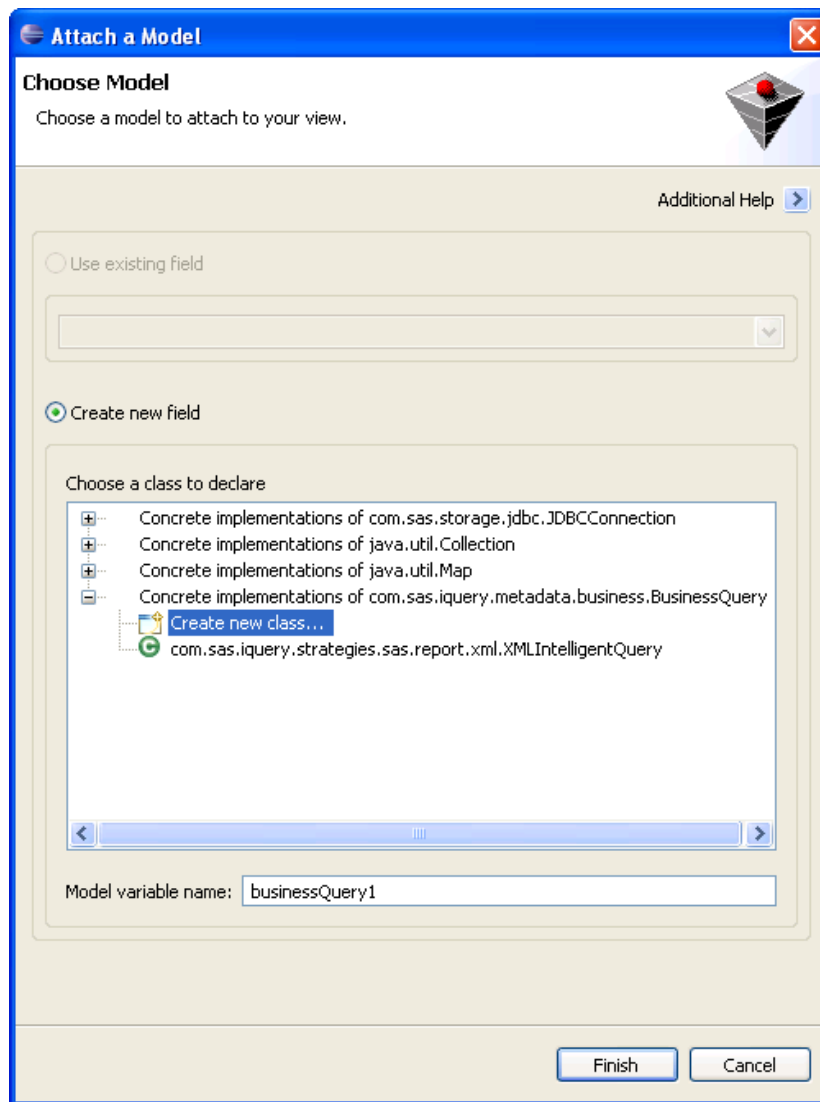
- 1 Place the cursor on a statement that contains a variable instantiation of a viewer type (for example, `javax.swing.JList`). The Quick Assist will not function if you place the cursor on a static method.
- 2 Press CTRL+1.

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**.



The Attach a Model dialog box is displayed.



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**:

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

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

b Enter the **Model variable name**.

This variable name is used in the declaration in the source file. The variable will always be declared in the topmost enclosing class.

A default variable name is supplied, based on the currently selected class. If you modify this name, it will no longer change when you select a different class.

- c Click **Finish** to add an initialization method to the source file with statements that instantiate the appropriate type for this variable (multiple fields share 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 are pieces of source code that you can configure and insert to reduce the amount of typing you do. Snippets are a robust form of the Eclipse Editor templates.

Enabling SAS Snippets

To enable snippets, follow these steps:

- 1 Select **Window ► Show View ► Other**.
- 2 Expand **Basic**.
- 3 Select **Snippets**.
- 4 Click **OK**.

The **Snippets** tab appears.

Inside the **Snippets** tab, click on **SAS Snippets** to see the available snippets.

Using SAS Snippets

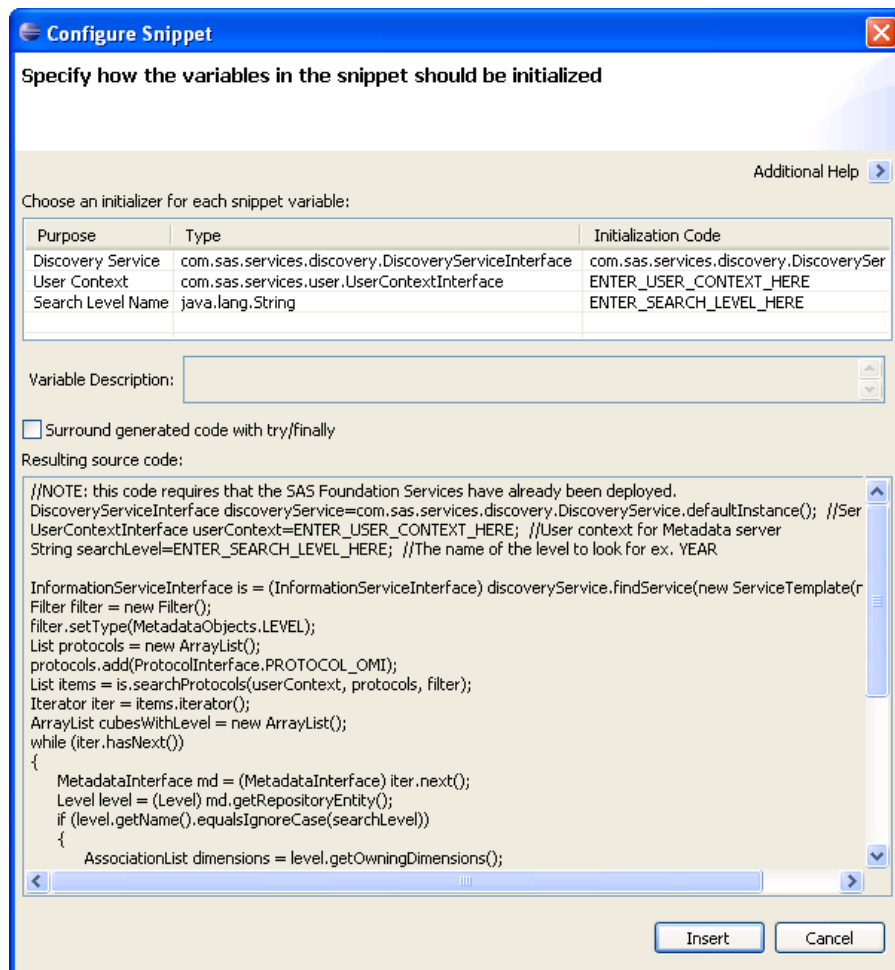
To use a snippet, follow these steps:

- 1 Place your cursor inside a method body where you want to insert the snippet.
- 2 Double click on the snippet you want to insert.

The Configure Snippet dialog box appears.

- 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.



The **Purpose** and **Type** of each variable is listed, and the **Initialization Code**, which defines how the variable is assigned. The **Initialization Code** can be set to one of three values:

- **ENTER_<PURPOSE>_HERE**

This illegal value is a string that will not compile so that you are forced to fix it. For example, **ENTER_HOST_NAME_HERE**.

- Arbitrary text you enter.

You can enter any text, method call, or other complex expression for the **Initialization Code**. Remember to include quotes and escape characters, if necessary.

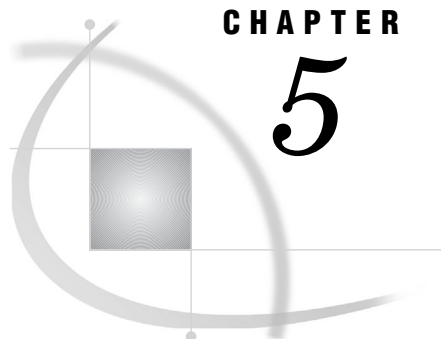
- A variable from your code.

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 you have selected is used throughout the snippet.

If you select the **Surround generated code with try/finally** check box, the generated code is enclosed in a **try/finally** block. This is useful if you want to handle the declared exceptions in the current method.

4 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.



CHAPTER

5

Manual Operations for a Project's SAS Repository

<i>Overview of Manual Operations for a Project's SAS Repository</i>	67
<i>Identifying Dependent Jars</i>	68
<i>Removing Jars from the Classpath</i>	69
<i>Changing the Order of Jars in the Classpath</i>	69
<i>Adding New Jars to the Classpath</i>	69
<i>Adding Dependent Jars</i>	70
<i>Specifying the Current Versions of Jars</i>	70
<i>Removing Version Restrictions</i>	71
<i>Selecting Jar Versions</i>	71
<i>Listing the Classpath Jars</i>	72
<i>Listing Jar Relationships</i>	72
<i>Finding a Class in a Jar</i>	73
<i>Changing Default Classes for SAS Java Projects</i>	74

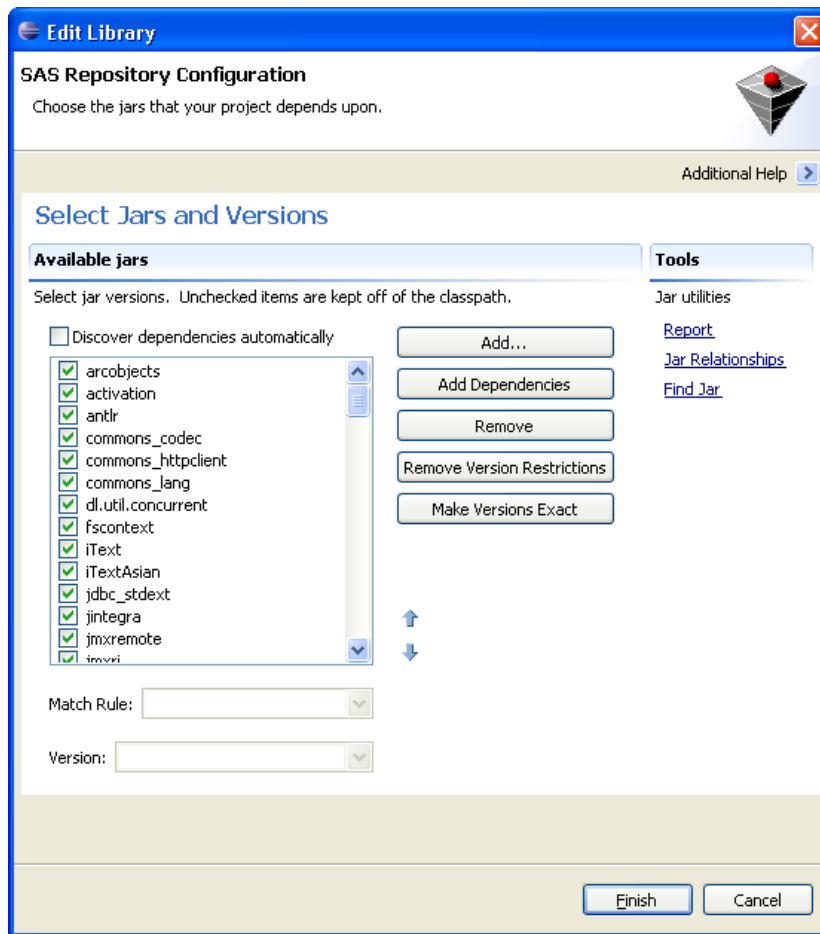
Overview of Manual Operations for a Project's SAS Repository

The following sections describe manual operations that you can perform on a project's SAS Repository. All operations are performed from the Edit Library dialog box.

To access the Edit Library dialog box:

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

The Edit Library dialog box appears.



Identifying Dependent Jars

To include on the classpath the dependencies of all checked jars, 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 particular versions of each of the dependency 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. The unchecked jar will not be 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; at compile time updates to the SAS Versioned Jar Repository might change the jars your project requires.

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 Kit. 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 **Configure**.
The Edit Library dialog box appears.
- 3 Select the jar(s) to remove and click the **Remove** button.

Note that using the Organize SAS Imports action will add a removed jar to the classpath (see “Using the Organize SAS Imports Action” on page 61). In contrast, unchecking a jar in the **Available jars** list (see “Identifying Dependent Jars” on page 68) means that the SAS Imports command will 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 **Configure**.
The Edit Library dialog box appears.
- 3 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:

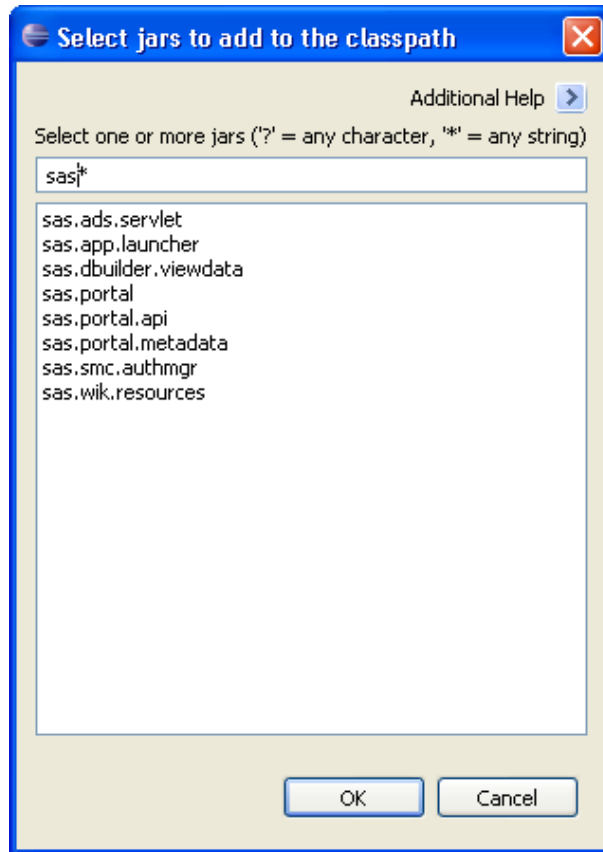
- 1 Expand the project in the Package Explorer.
- 2 Right-click **SAS Repository** and select **Configure**.
The Edit Library dialog box appears.
- 3 Click the **Add** button.

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

To determine if there are missing jars that the project's current jars require, follow these steps:

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

The Edit Library 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 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, follow these steps:

- 1 Expand the project in the Package Explorer.

- 2 Right-click **SAS Repository** and select **Configure**.

The Edit Library dialog box appears.

- 3 Select the jar(s) and click **Make Versions Exact**.

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 **Configure**.

The Edit Library dialog box appears.

- 3 Select the jar and click **Remove Version Restrictions**.
-

Selecting Jar Versions

If you want to use a version of a jar other than the latest version, you must specify it. If there is no version specified for a jar, then the latest version of the jar is used.

You can have only one version of a jar in your classpath. 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.

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 **Configure**.

The Edit Library dialog box appears.

- 3 Select the jar(s) 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.

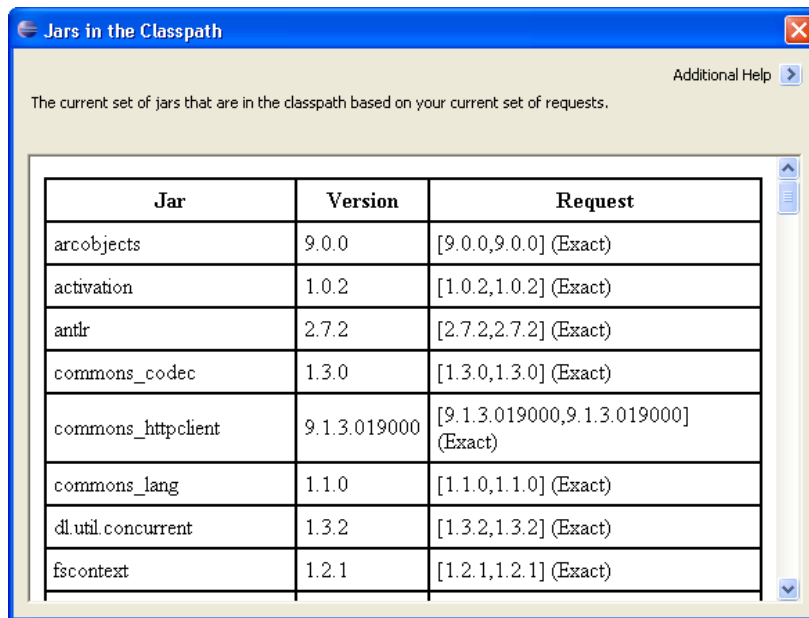
- 5 Select the **Version**.

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. Keep in mind that the jars themselves might have their own version requirements for dependent jars.

Listing 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 **Configure**.
The Edit Library dialog box appears.
- 3 Click **Report**.



Jar	Version	Request
arcobjects	9.0.0	[9.0.0,9.0.0] (Exact)
activation	1.0.2	[1.0.2,1.0.2] (Exact)
antlr	2.7.2	[2.7.2,2.7.2] (Exact)
commons_codec	1.3.0	[1.3.0,1.3.0] (Exact)
commons_httpclient	9.1.3.019000	[9.1.3.019000,9.1.3.019000] (Exact)
commons_lang	1.1.0	[1.1.0,1.1.0] (Exact)
dl.util.concurrent	1.3.2	[1.3.2,1.3.2] (Exact)
fscontext	1.2.1	[1.2.1,1.2.1] (Exact)

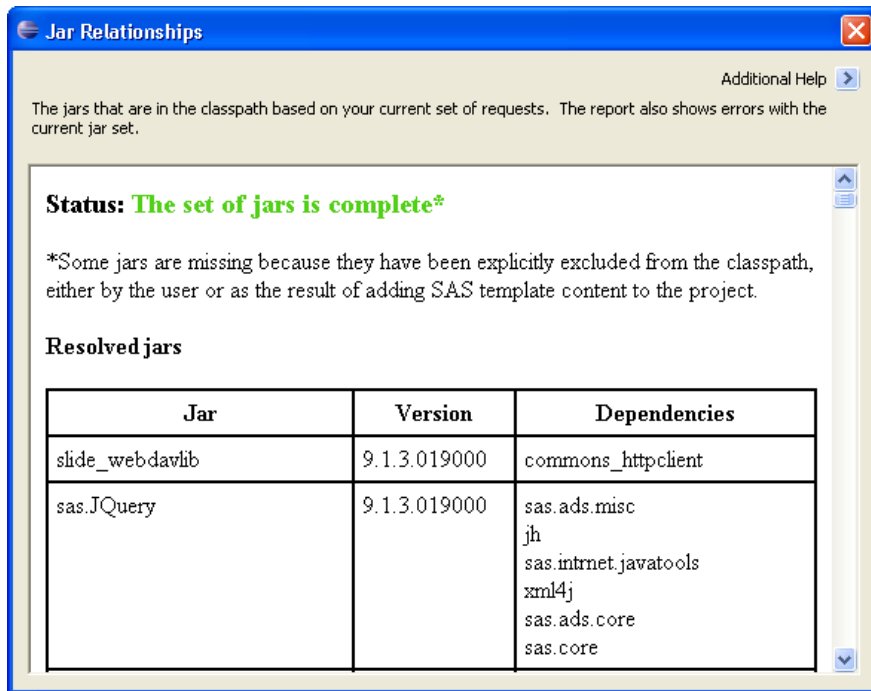
Listing Jar Relationships

To generate a report on 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 **Configure**.
The Edit Library dialog box appears.
- 3 Click **Jar Relationships**.

Status text across the top of the Jar Relationships dialog box indicates if the set of jars is complete, or the type of error found. 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 but are not currently included. The jar IDs are listed, along with the name of the jar that directly requested them.
- **Unresolved jars** – lists jars that could not be resolved. Displays the version request string and the likely error.



Finding a Class in a Jar

To determine which of your project's jars contains a class, and to potentially add that jar to the SAS Versioned Jar Repository, follow these steps:

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

The Edit Library dialog box appears.

- 3 Click **Find Jar**.

The Find Jar dialog box appears.

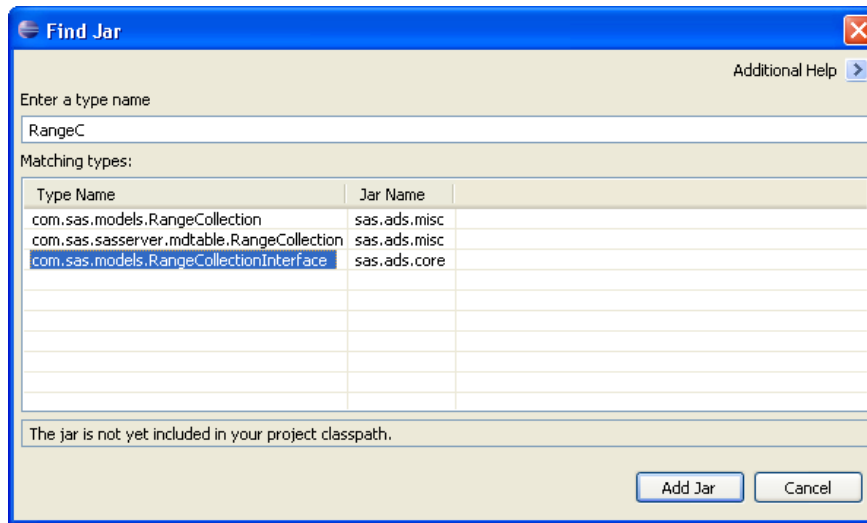
- 4 Enter the local name of a class (not the package).

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

- 5 Select the class you want from the list.

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

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

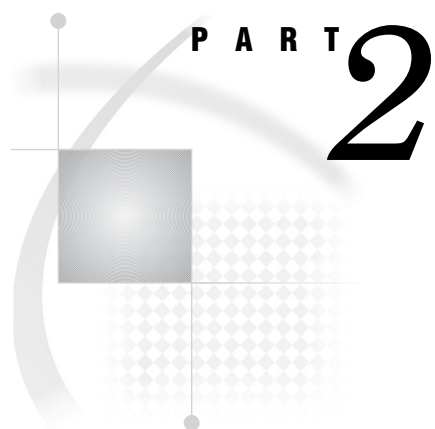


Changing Default Classes for SAS Java Projects

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:

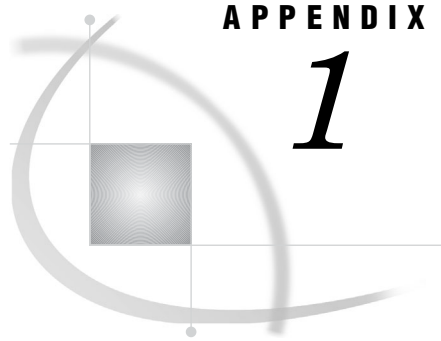
- 1 Select **Window ► Preferences**.
- 2 Expand **SAS AppDev Studio**, and select **SAS Project dependencies**.
- 3 Change the included jars just as you would for a project.
- 4 Click **OK**.

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



Appendices

Appendix 1 **Additional Tasks and Resources** 77



APPENDIX

1

Additional Tasks and Resources

<i>Understanding SAS Web Application Projects</i>	77
<i>Restoring SAS Versioned Jar Repository Dependencies to SAS Web Application Projects</i>	78
<i>Standard SAS Filter and Servlet Declarations and Mappings</i>	79
<i>Importing the ADS Local Services into a SAS Metadata Repository</i>	80
<i>Modifying the ADS Local Services</i>	81
<i>Creating a New Target Run Time</i>	82
<i>SAS Java Web Application Resources</i>	82
<i>VM Startup Arguments</i>	83
<i>Java Security Permissions</i>	83
<i>Foundation Services Context Listener Class</i>	84
<i>Logging Configuration</i>	85
<i>Examples HttpSessionBindingListener Class</i>	85
<i>JAAS Login Configuration</i>	85
<i>Local Services Metadata Properties</i>	85
<i>Remote Services Metadata Properties</i>	85
<i>Tomcat Context Configuration</i>	86
<i>Deployment Descriptor</i>	86
<i>Setting up Trusted Web Authentication for SAS Web Applications</i>	86
<i>Add a Web Domain User to the Metadata</i>	86
<i>Set the Domain of the Information Service</i>	87
<i>Set the Web Application Security Constraints</i>	87
<i>Define the Web User Role for the Web Application Server</i>	89
<i>Specifying Separate Java Source and Output Directories</i>	89

Understanding SAS Web Application Projects

In your project you can use classes or tag libraries other than those supplied with SAS AppDev Studio 3.2 Eclipse Plug-ins. For example, you can add Apache Struts or JavaServer Faces to your project, or put Java utility classes in a jar that will be included in a Web application's **WEB-INF/lib/** directory. First, however, you need to understand how SAS Web Application Projects build upon the Dynamic Web Project of the Web Tools Platform (versions 1.0.1 through 1.0.3).

Fundamentally, a SAS Web Application Project consists of two types of resources: static content from the SAS Web Infrastructure Kit (including the filter and servlet declarations), and SAS jars from a SAS Versioned Jar Repository.

The static content from the SAS Web Infrastructure Kit is copied into a project's Web content directory. And although this content is accessible within the project, it should not be edited.

In contrast, the SAS Versioned Jar Repository jars are programmatically added to a project as J2EE module dependencies, and are not immediately added to a project's **WEB-INF/lib/** directory.

To verify the jars that will be published to a project's **WEB-INF/lib/** directory, right-click the project and select **Project ► Properties**. Then select **J2EE Module Dependencies**. The SAS jars included in the list (even if unchecked) will be included in the Web application's **WEB-INF/lib/** directory when the Web application is published to a server or exported as a WAR file.

The jars from the SAS Versioned Jar Repository are all unchecked because the J2EE Module Dependencies properties page checkmarks only those jars it finds listed explicitly in the project's Java Build Path. Normally, J2EE dependencies are added to the Java Build Path. However, because the J2EE Module Dependencies page does not look inside classpath containers that are listed in the Java Build Path (such as the SAS Versioned Jar Repository), the jars within these classpath containers are not found. This causes the programmatically added SAS jars to be unchecked. Because of this limitation, you should not change the state of a check box for any SAS Versioned Jar Repository jar. Instead, you should manage the SAS Versioned Jar Repository jars manually (see Chapter 5, “Manual Operations for a Project's SAS Repository,” on page 67).

Because the J2EE Module Dependencies properties page does not work with classpath containers (in this case the SAS Versioned Jar Repository), if you add or remove jars using this page, even jars that are not part of the SAS Versioned Repository jars, all the SAS Versioned Repository jars might be removed from the J2EE dependencies list. The project will still build successfully, but the SAS jars will not be included when the Web application is published to a server or exported as a WAR file. Moreover, checkmarking the jars in an effort to include them would lead to duplicate classpath entry errors.

If the SAS Versioned Repository jars disappear from the J2EE Module Dependencies list, they are not automatically restored. Instead, you must restore them manually (see “Restoring SAS Versioned Jar Repository Dependencies to SAS Web Application Projects” on page 78).

The two parts of a SAS Web Application Project, the static content from the SAS Web Infrastructure Kit and the SAS Versioned Jar Repository jars, are added to the project using the Web Tools Facet support. All SAS Web Application Projects will have the **SAS Web Module with WIK** item checked on the Project Facets page of the Project properties. For a Dynamic Web Project, **SAS Web Module with WIK** will be unchecked. Adding the SAS Web Module with WIK facet to a Dynamic Web Project converts it into a SAS Web Application Project.

However, the SAS Web Module with WIK facet does not add to the project the standard SAS filter and servlet declarations and mappings. These are added when you add to the project a template for one of the SAS Web Application Examples. If you do not add such a template, you must add the SAS filter and servlet declarations manually. For more information about adding these elements manually, see “Standard SAS Filter and Servlet Declarations and Mappings” on page 79.

For the latest information about SAS Web Application Projects, see <http://support.sas.com/rnd/appdev/doc/SASWebAppDev.html>.

Restoring SAS Versioned Jar Repository Dependencies to SAS Web Application Projects

Toggling the inclusion of any project jar (for example, **sas.swing.remote**), causes the re-evaluation and restoration of the project's SAS Versioned Jar Repository dependent entries.

To restore the SAS Versioned Jar Repository entries in the J2EE Module Dependencies, follow these steps:

- 1 Open the SAS Repository Configuration dialog box:
 - a In the Package Explorer or Project Explorer view, expand the project.
 - b Right-click **SAS Repository** and select **Configure**.
- 2 Uncheck the first jar in the list and click **Finish**.
- 3 Reopen the SAS Repository Configuration dialog box and select the check box for the first jar. Click **Finish**.

Standard SAS Filter and Servlet Declarations and Mappings

The SAS Java components require that specific XML filter and servlet declarations and mappings be included in the web.xml of SAS Web Applications. If you create a Web Application project using one of the templates for SAS Web Application Examples, these declarations and mappings are provided for you. However, if you are not using a provided template, then these XML entries must be added manually.

For Servlet 2.3 Web applications, these XML elements must appear in the proper order. The <filter> and <filter-mapping> elements should appear after all elements except <listener> and <servlet> elements. The <servlet> and <servlet-mapping> elements should appear after <listener> elements, but before the <welcome-file-list> element. If you have Eclipse Web Tools Platform XML Validation enabled (the default), an error will occur if the elements are added out of order.

There are other elements, but describing them all here is impractical. The full DTD is available in the Servlet 2.3 and 2.4 specifications, which can be downloaded from <http://java.sun.com/products/servlet/download.html#specs>.

The required filters:

```
<filter>
    <filter-name>CharacterEncodingFilter</filter-name>
    <filter-class>com.sas.servlet.filters.CharacterEncodingFilter
    </filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>CharacterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

The required servlets:

```
<servlet>
    <servlet-name>MethodInvocationServlet</servlet-name>
    <servlet-class>
        com.sas.servlet.util.MethodInvocationServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>SelectorServlet</servlet-name>
    <servlet-class>
        com.sas.servlet.tbeans.dataselectors.SelectorServlet</servlet-class>
</servlet>
```

```

<servlet>
    <servlet-name>StreamContentServlet</servlet-name>
    <servlet-class>
        com.sas.servlet.util.StreamContentServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>VisualDataExplorerServlet</servlet-name>
    <servlet-class>
        com.sas.servlet.util.VisualDataExplorerServlet</servlet-class>
</servlet>
<servlet>
    <servlet-name>Model2ExampleControllerServlet</servlet-name>
    <servlet-class>
        servlets.Model2ExampleControllerServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>MethodInvocationServlet</servlet-name>
    <url-pattern>/MethodInvocationServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>SelectorServlet</servlet-name>
    <url-pattern>/SelectorServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>StreamContentServlet</servlet-name>
    <url-pattern>/StreamContentServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>VisualDataExplorerServlet</servlet-name>
    <url-pattern>/VisualDataExplorerServlet</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>Model2ExampleControllerServlet</servlet-name>
    <url-pattern>/Model2Example</url-pattern>
</servlet-mapping>

```

Importing the ADS Local Services into a SAS Metadata Repository

The templates for both the Information Map and Web Report Viewer categories of the SAS Web Application examples make use of local SAS Foundation Services. By default, those templates deploy the ADS Local Services from a SAS Metadata Repository. But before you can use those templates, the ADS Local Services must be manually imported into your SAS Metadata Repository using the SAS Management Console.

The ADS Local Services deployment file is named **sas_services_ads_local_omr.xml**, and is located in the **AppDevStudio** directory that is in the SAS AppDev Studio Eclipse Plug-ins installation directory. The ADS Local Services deployment file must be accessible to the SAS Management Console.

To import the ADS Local Services into your SAS Metadata Repository, follow these steps:

- 1 Start the SAS Management Console and open a connection to the SAS Metadata Repository running on the system that contains your SAS AppDev Studio Eclipse Plug-ins installation.

- 2 Right-click the **Foundation Services Manager** and select **Import Service Deployment**.

The Import Service Deployment dialog box appears.

- 3 Click **Add**.
- 4 Navigate to the directory that contains the `sas_services_ads_local_omr.xml` file.

Typically, this file is located in the AppDevStudio directory under the root installation directory for SAS AppDev Studio 3.2 Eclipse Plug-ins. For example:
C:\Program Files\SAS\SASAppDevStudio\3.2\AppDevStudio.

- 5 Select the file named `sas_services_ads_local_omr.xml` and click **Open**.
- 6 Click **OK**.

CAUTION:

You might need to modify the ADS Local Services that you just imported. In the ADS Local Services deployment file just imported, both the Platform Information Service and the Platform User Service contain a host name that is configured by default to be the machine name of the system on which SAS AppDev Studio 3.2 Eclipse Plug-ins was installed. These two services also assume that the SAS Metadata server is using port 8561. As a result, you must modify the ADS Local Services if you want to do any of the following:

- run the SAS Metadata server on a system different than where SAS AppDev Studio 3.2 Eclipse Plug-ins is installed.
- replace the machine name with a fully qualified DNS name to better simulate a production environment.
- use a SAS Metadata server port other than 8561.

△

Modifying the ADS Local Services

The AppDev Studio 3.2 Eclipse Plug-ins installer assumes that your SAS Metadata Repository server is on the same physical system as the AppDev Studio plug-ins. Likewise, the ADS Local Services deployment contains the host of the SAS Metadata Server in the configuration for the Platform Information Service and Platform User Service. When SAS AppDev Studio 3.2 Eclipse Plug-ins is installed, this host is set to be your machine name.

You must modify the deployment metadata in the ADS Local Services if any of the following are true:

- You deploy the ADS Local Services to a SAS Metadata Repository on a system other than the one on which SAS AppDev Studio 3.2 Eclipse Plug-ins is installed.
- You want to change the machine name to a fully qualified DNS name to better simulate a production environment.
- Your SAS Metadata server is using a port other than 8561.

To modify the deployment metadata in the ADS Local Services, import the ADS Local Services into the SAS Metadata Repository, and then follow these steps:

- 1 In the SAS Management Console tree, expand **Foundation Services Manager**, **ADS Local Services**, and **BIP Core Services**.
- 2 Right-click the **Platform Information Service** under BIP Core Services and select **Properties**.
- 3 Switch to the **Service Configuration** tab and click **Edit Configuration**.

- 4 With **Foundation** selected in the Information Repositories list, click **Edit**.
- 5 Enter the **Host** and **Port** field values. A host name in the **Description** field is optional.
- 6 Click **OK**.
- 7 Right-click the **Platform User Service** under BIP Core Services and select **Properties**.
- 8 Switch to the Service Configuration tab and click **Edit Configuration**.
- 9 Select the **Profiles** tab.
- 10 With the global profile selected in the Applications list, click **Edit**.
- 11 Update the host and port found in the **Domain URL** field.
- 12 Click **OK**.

Creating a New Target Run Time

To create a new target run time, follow these steps:

- 1 Select **Window ► Preferences** from the main Eclipse window.
- 2 Expand the **Server** branch and select the **Installed Runtimes**.
This will display a table of all the installed server runtimes that Eclipse knows about.
- 3 Click **Add**.
- 4 Expand the **Apache** branch and select **Tomcat v4.1 Server**.
- 5 Click **Next**.
- 6 Change the **Name** field to *SAS local Tomcat v4.1*.
- 7 Enter the **Tomcat installation directory**.
- 8 In the JRE combo box select **SAS private JRE 1.4.2_05**.
- 9 Click **Finish**.

SAS Java Web Application Resources

A template for a SAS Java Web Application example consists of all the files and resources needed to implement that application (typically a servlet and JSP files, and other support resources, like settings). If a second template is added to the project, some of the resources needed by the second template might already exist in the project. In such cases, only the files that are both unique to that template and are also needed are added by the second SAS Web Application template. These unique resources are called the *primary* resources of a template. For example, the primary resource of the *Context Listener* templates is the Context Listener Java file.

The majority of the templates for the SAS Web Application Examples use the SAS Foundation Services or some form of connection to SAS. When you add these templates to a project, they ensure that the required resources are present in the SAS Web Application project, and create or alter those resources as necessary.

The Foundation Services Support templates also create or alter the required resources. However, they provide more user control over the name and location of the primary resources the template requires. For example, when using a SAS Web Application template you cannot specify the name of the context listener (it is always `FoundationServicesContextListener`), or its package. In contrast, adding the same

context listener with the Foundation Services Support templates enables you to specify its name and package.

VM Startup Arguments

Provided in **launchParameters.txt**.

The SAS Java Web Application templates use this file to provide JVM arguments to the Tomcat Server's Java launch configuration. This file is added to the root of the SAS Web Application Project. For details on its use, see "Configuring the Web Application Server Launch Settings" on page 16.

Note that when you add a template to a project, the **launchParameters.txt** file might be rewritten, discarding any customizations you have made. If you customize the file in any way, such as altering arguments or adding additional arguments, save a copy of the file or just the additions elsewhere in the project. This way you can easily add your customizations to the new **launchParameters.txt** file.

Java Security Permissions

Provided in **java.webapp.policy**.

This file, located in the root folder of the project, is a full Tomcat 4.1.18 Java policy file, containing permissions for the Tomcat server as well the Web application. If you are running only one Web application on the server, you can use the entire file as the **catalina.policy** file. If you are running multiple Web applications on the server, you need to merge the permissions for each Web application into the **catalina.policy** file. For additional information on using this file, see the "Adding Security Policies to the Web Application Server Policy File" on page 15.

To add the **java.webapp.policy** file to a project, add to the project either of the following Foundation Services Support templates:

- Tomcat Server Policy File with Security Manager
- Tomcat Security Policy File without Security Manager

There are two versions of the **java.webapp.policy** file: one that is used with a security manager enabled, and one that is used without a security manager enabled. All of the templates for SAS Web Application Examples create or modify the **java.webapp.policy** file to be the version that works with a security manager enabled.

The version of the **java.webapp.policy** file that works without a security manager is available only from the *Tomcat Security Policy File without Security Manager* template. This version of the policy file is provided because even if you run your Web application without a security manager, there are hard-coded permission checks in Foundation Services code that require the permissions that are provided by this version of the file. This file contains only these required permissions, and does not contain the permissions required if a security manager is enabled.

The version of the **java.webapp.policy** file that works with a security manager can be used with or without a security manager because the file includes the permissions needed by the Foundation Services code (among many others).

As a result, if you try to add the *Tomcat Security Policy File without Security Manager* template to a project (and the corresponding **java.webapp.policy** file that works without a security manager), and the project already has a version of the **java.webapp.policy** file that works with a security manager, the **java.webapp.policy** file is not modified because the existing policy file will work.

If you run the Tomcat server without the **Enable security configuration** setting checked, you must manually add the **java.security.policy** System property to the Tomcat launch configuration file. The property value must be the absolute path to the

`java.webapp.policy` file in your project, or to a policy file containing appropriate SAS Web application permissions. Although possible, running Tomcat without the **Enable security configuration** setting checked is not supported because SAS AppDev Studio Eclipse 3.2 Plug-ins expects the Tomcat server to have the **Enable security configuration** option turned on.

The **Enable security configuration** check box is in the Tomcat server configuration editor, configurable from within Eclipse.

Foundation Services Context Listener Class

Provided in `listeners.FoundationServicesContextListener`.

This Java file implements the `ServletContextListener` interface class, and is used by SAS Web applications to handle lifecycle issues related to SAS Foundation Services. It is added to the Java Source Folder specified when you add the template.

Although there is only one `listeners.FoundationServicesContextListener` file, it can be used three different ways, depending on the template used to add it to a project. The templates, depending on their requirements, can create or alter the file to match one of three functional levels:

- 1 The `FoundationServicesContextListener` handles the destruction of default services during SAS Web application shutdown. Typically, these services are default logging services.

To add this level of functionality to the `FoundationServicesContextListener` class, add to the project the *Context Listener for Default Services* template found in the Foundation Services Support category

- 2 The `FoundationServicesContextListener` handles the deployment of local services at SAS Web application startup, and their destruction at shutdown.

To add this level of functionality to the `FoundationServicesContextListener` class, add to the project the *Context Listener for Local Services* template found in the Foundation Services Support category.

- 3 The `FoundationServicesContextListener` handles the deployment of local and remote services at SAS Web application startup, and their destruction at shutdown.

To add this level of functionality to the `FoundationServicesContextListener` class, add to the project the *Context Listener for Remote & Local Services* template found in the Foundation Services Support category.

When necessary, a SAS Web application template can overwrite the `FoundationServicesContextListener` file with one of a higher level of functionality. Templates never downgrade the file to a lower functional level. If the file is at higher level than what the template requires, the file is not altered.

When the `FoundationServicesContextListener` class is used to deploy services, it uses the **Local Services Metadata Properties** and/or the **Remote Services Metadata Properties** files to specify the services to deploy. The locations of these files within the SAS Web application are communicated to the `FoundationServicesContextListener` using context parameters in the SAS Web application's `web.xml` file, as described in the comments near the top of the `FoundationServicesContextListener` file.

For more information on these service files, see “Local Services Metadata Properties” on page 85 and “Remote Services Metadata Properties” on page 85.

At all functional levels, the `FoundationServicesContextListener` can configure the Logging Service by defining a context parameter that specifies the location of the desired configuration file. For details, see “Logging Configuration” on page 85.

The default file name for the class is `FoundationServicesContextListener.java`, and its default location is the `listeners` package in the Java source folder of the project.

The Foundation Services Support templates enable you to choose the name and package for this class.

Logging Configuration

Provided in **LoggingServiceConfig.xml**.

To create this resource, add to the project the Logging Configuration template found in the Foundation Services Support category.

The template creates the **LoggingServiceConfig.xml** file in
 /<SAS Web App Project>/WebContent/WEB-INF/.

This template also adds to the **web.xml** file a context parameter that is needed by the Foundation Services Context Listener Class to make use of the **LoggingServiceConfig.xml** file. The **LoggingServiceConfig.xml** file can be used as a starting point to customize the locally deployed Logging Service.

Examples HttpSessionBindingListener Class

Provided in **listeners.ExamplesSessionBindingListener**.

This Java file implements the **HttpSessionBindingListener** interface. Several of the templates in the SAS Web Application Examples category include this file to handle resource cleanup when HTTP sessions expire or the SAS Web application is shut down. The file is added to the Java Source Folder specified when adding the template. Additional details about this class are provided in the comments in the **listeners.ExamplesSessionBindingListener** file.

JAAS Login Configuration

Provided in **login.config**.

This Java Authentication and Authorization Service (JAAS) login configuration file is used to authenticate against a SAS Metadata Server, the only authentication supported by the SAS Web Application templates. The **login.conf** file is placed in the root of the SAS Web Application Project. This file can be added to the project using the JAAS Login Configuration File template found in the Foundation Services Support category.

This file contains references to the Metadata Server host, port, repository, and authentication domain. It does not contain a password.

Local Services Metadata Properties

Provided in **sas_metadata_source_omr.properties**.

This properties file is used by the Foundation Services **ServletContextListener** to deploy local services. It is placed in

/<SAS Web App Project>/WebContent/WEB-INF/conf/. The **ServletContextListener** finds this file within the SAS Web application content using a context parameter in the **web.xml** file. For details, see the “Foundation Services Context Listener Class” on page 84.

This file contains references to the Metadata Server host, port, repository, user, and password.

Remote Services Metadata Properties

Provided in **sas_remote_metadata_source_omr.properties**.

This properties file is used by the Foundation Services **ServletContextListener** to deploy remote services. The file is placed in `/<SAS Web App Project>/WebContent/WEB-INF/conf/`. The **ServletContextListener** finds this file within the SAS Web application content using a context parameter in the **web.xml** file. For details, see the “Foundation Services Context Listener Class” on page 84.

This resource file contains references to the Metadata Server host, port, repository, user, and password.

Tomcat Context Configuration

Provided in **context.xml**.

This file contains information for configuring the project’s Web application on Tomcat. Specifically, it contains the information needed to disable the default Tomcat behavior of serializing HTTP session data on shutdown, and restoring the session data on startup. This resource is placed in `/<SAS Web App Project>/WebContent/META-INF/`.

SAS Web applications typically make use of connections to SAS servers that are held for a user in their HTTP session. Serializing and deserializing the Java side of a connection to a SAS server does not restore the connection to the SAS server. Therefore, the SAS server connections (Java objects) are marked as non-serializable. In Tomcat, attempts to serialize and deserialize these non-serializable objects can cause exceptions that can interfere with startup and shutdown, eventually making the Web application malfunction due to corrupt session data. To avoid this, the **context.xml** file contains a context configuration to disable the Tomcat session serialization. The Tomcat server support in the Eclipse Web Tools includes this context configuration information when adding a Web project to the server.

Deployment Descriptor

Provided in **web.xml**.

This file contains declarations that are both added to the file and altered by the templates. The templates do not create the file, they only make changes to it.

The SAS Web Application Examples involving any of the Information Map or Report Viewer templates specify Metadata Server authentication domain, user ID, and password in servlet init parameter settings that serve as the user credentials for the SAS Web Application. These servlet init parameters are stored in the **web.xml** file.

Setting up Trusted Web Authentication for SAS Web Applications

The templates in the SAS Web Application Examples category are configured by default to use host authentication. To establish trusted Web authentication for a SAS Web Application, follow these steps:

- 1 Add a Web domain user to the metadata.
- 2 Set the Domain of the Information Service.
- 3 Set the Web Application Security Constraints.
- 4 Define the Web User Role for the Web application server (assumed to be Tomcat).

Add a Web Domain User to the Metadata

To add a Web domain user identity, follow these steps:

- 1 Log into SAS Management Console and connect to the SAS Metadata Server where you imported, or plan to import, the ADS Local Services.
- 2 Click on **User Manager**.
- 3 Right-click **SAS Demo User** and select **New ► User**.
- 4 Select the **General** tab.
- 5 Enter a name for the user, such as *SAS Web Demo User*.
- 6 Select the **Logins** tab.
- 7 Click **New**.
- 8 Enter a user ID for the SAS Web Demo User (such as, *sasWebDemoUser*). Also specify the Authentication Domain as *web*.

You do not need to enter or confirm a password. Because this authentication is Web-based rather than host-based, do not precede the user ID with a hostname. If you do not see *web* in the list of Authentication Domains, add it using the **New** button next to the Authentication Domain field.

- 9 Click **OK**.

Set the Domain of the Information Service

This section assumes that you have imported the ADS Local Services service deployment. See “Importing the ADS Local Services into a SAS Metadata Repository” on page 80 for more information on importing services.

To set the domain of the Information Service, follow these steps:

- 1 Log into SAS Management Console and connect to the SAS Metadata Server where you imported the ADS Local Services.
- 2 Select **Foundation Services Manager**.
- 3 (Optional but recommended.)

If you want to isolate the service deployment for trusted authentication from the host authentication, duplicate the host authentication resource and give the duplicate a new name:

 - a Right-click ADS Local Services and select **Duplicate Service Deployment**.
 - b Enter a name for the duplicate and click **OK**.
- 4 In the tree, expand **ADS Local Services Trusted** and then **BIP Core Services**.
- 5 Right-click **Platform Information Service** and select **Properties ► Service Configuration ► Edit Configuration ► Repositories**.
- 6 In the Information Service Configuration dialog box, select the **Repositories** tab.
- 7 Select your repository from the list.
- 8 Click **Edit**.
- 9 Change the **Domain** field to *web*.
- 10 Click **OK** in the Edit Information Service Repository dialog box.
- 11 Click **OK** in the Information Service Configuration dialog box.

Set the Web Application Security Constraints

This section assumes you are using one of the information map templates in the SAS Web Application Examples category, such as Information Map Viewer Servlet (**InformationMapViewExample**) or Information Map OLAPTableView Servlet (**OLAPTableViewExample**).

Define your security constraints in the **web.xml** file. The **web.xml** file is located in **/<SAS Web App Project>/WebContent/WEB-INF/**.

- 1 Add the following to your **web.xml** file before the closing `<web-app>` tag:

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Success</web-resource-name>
        <url-pattern>*/</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>webuserRole</role-name>
    </auth-constraint>
</security-constraint>

<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>default</realm-name>
</login-config>

<security-role>
    <role-name>webuserRole</role-name>
</security-role>
```

- 2 In the **web.xml**, change the metadata-domain initparam value from *DefaultAuth* to *web*.
- 3 Modify the `newUser` object in your Controller Servlet to include the remote user. Open your **xxxxControllerServlet.java** file in Eclipse and change

```
UserContextInterface userContext =
    userService.newUser(username, password, domain);
```

to

```
UserContextInterface userContext =
    userService.newUser(request.getRemoteUser(), "", domain);
```

- 4 (Optional)

Clean up the unused code references by removing the following two lines of code from the ControllerServlet:

```
String username = sc.getInitParameter("metadata-userid");
String password = sc.getInitParameter("metadata-password");
```

Remove from the **web.xml** file the corresponding user ID and password references:

```
<init-param>
    <param-name>metadata-userid</param-name>
    <param-value>sasguest</param-value>
</init-param>
<init-param>
    <param-name>metadata-password</param-name>
    <param-value>{sas001}U0FTcHcx</param-value>
</init-param>
```

Define the Web User Role for the Web Application Server

- 1 In Eclipse, expand the Tomcat server instance under the Servers project in the Project Explorer (the one your project is tied to).
- 2 Double-click the **tomcat-users.xml** file and add definitions similar to the following:

```
<role rolename="webuserRole"/>
<user username="sasWebDemoUser" password="sasdemo1" roles="webuserRole"/>
```

You will be prompted at run time (once per browser session) for this information when you access the Web application's URL.

- 3 Add the following login module information to your **login.config** file (located under your project root directory). Modify this information to reflect your hostname, port, repository, SAS Trusted User ID, and password.

Setting the debug option to true can help troubleshoot a problem.

```
PFS {
    com.sas.services.security.login.TrustedLoginModule optional "debug"="false"
    "host"="<hostname>.na.sas.com"
    "port"="8561"
    "repository"="Foundation"
    "domain"="web"
    "trusteduser"="<unqualified servername>/sastrust
    "trustedpw"="SASpw1";
};
```

Specifying Separate Java Source and Output Directories

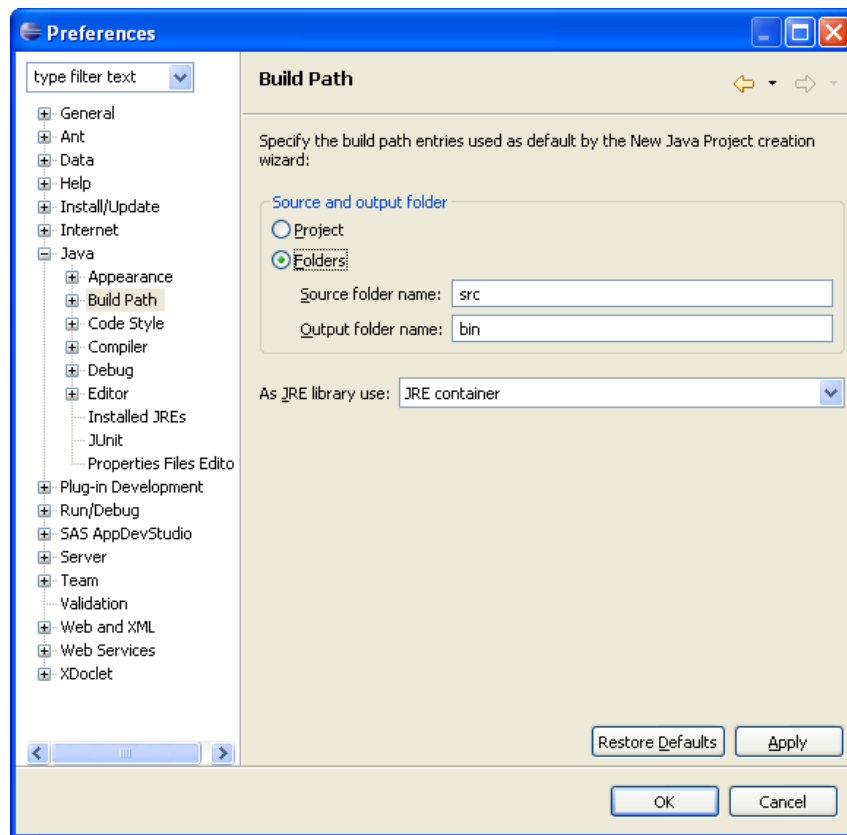
To avoid jar packaging warnings and to simplify development tasks, you should specify separate source and output directories rather than accept the Eclipse default setting of using the same folder.

To specify separate source and output directories as the default for Java projects, follow these steps:

- 1 Select **Window ► Preferences**.
- 2 Expand **Java** and click on **Build Path**.
- 3 Click on **Folders**.
- 4 Specify the **Source folder name** and **Output folder name**.

These folders are relative to the root of the project directory.

- 5 Click **OK**.



Index

A

ADS Local Services
 importing 80
 modifying 81
 API documentation 59

C

classes
 changing default for SAS Java Projects 74
 Context Listener class 84
 Context Listener templates 51

D

Data Integration Studio plug-ins 27
 DataBeans 29
 dependent jars
 identifying 68
 deployment descriptor 86
 using 10
 documentation
 accessing 59

E

Eclipse projects
 upgrading to a SAS project 8
 exporting
 SAS Java projects as jars 9
 SAS Web applications as a WAR file 10

F

filters 79
 Foundation Services
 Context Listener class 84

H

help
 accessing 19
 HttpSessionBindingListener 85

I

import statements
 adding 60
 Organize SAS Imports action 61
 Repository Quick Fixes 60
 importing
 webAF Java projects 7
 webAF Java Web application projects 8
 Information Map servlet templates
 completing Default Servlets 44
 completing TableView Servlets 44
 completing Viewer Servlets 43
 preparing the server to run 44
 running 45
 using 38
 Information Service
 setting domain 87

J

JAAS
 login configuration 85
 jars
 changing order on classpath 69
 finding a class in 73
 listing classpath jars 72
 listing jar relationships 72
 removing from classpath 69
 removing version restrictions 71
 specifying current version 70
 specifying version used 71
 Java webapp
 security 83
 Javadoc 59
 configuring with Eclipse 60
 JDBC Servlets 33
 completing 37
 preparing server to run 37
 running 37

L

launch parameters file 83
 launch settings
 configuring for the Web application server 16
 Local Services
 metadata properties 85

logging configuration 85
 Logging Configuration template
 using 56

M

metadata properties
 Local Services 85
 Remote Services 85
 model/viewer connections 61
 supported viewers 62
 using the model/viewer connection Quick Assist 62

O

Organize SAS Imports action 61
 output directory
 specifying 89

P

PAR files
 creating from portlets 23
 portlets
 packaging into PAR files 23

Q

Quick Assists
 enabling 62

R

Remote Services
 metadata properties 85
 Report Viewer servlet
 completing 50
 preparing server to run 51
 running 51
 Report Viewer servlet template 45
 Repository
 SAS Repository 3
 SAS Versioned Jar Repository 3

S

- SAS ID Portal Portlets 20
- SAS Java Projects
 - creating 5
- SAS Management Console Plug-ins 23
- SAS Metadata
 - adding a web domain user to 86
- SAS Metadata Repository
 - importing ADS Local Services 80
- SAS Repository 3
- SAS Versioned Jar Repository 3
 - copying 10
 - restoring dependencies to 78
- SAS Web applications
 - creating 4
 - exporting as a WAR file 10
 - restoring SAS Versioned Jar Repository dependencies 78
 - running 17
 - understanding 77
 - web authentication 86
- security
 - permissions 83
 - Web application constraints 87
- security policies
 - adding to the policy file 15
- server policy file 15
- servlet declarations and mappings 79
- snippets 64

- source directory
 - specifying 89

T

- target run-time
 - creating 82
- templates
 - adding to an existing project 14
 - Context Listener 51
 - Data Integration Studio plug-in 27
 - DataBean 29
 - help for 19
 - Information Map Servlet 38
 - JDBC servlet 33
 - Logging Configuration 56
 - Report Viewer servlet 45
 - SAS ID Portal Portlet 20
 - SAS Management Console plug-in 23
 - Web application 13
- Tomcat
 - context configuration 86

U

- upgrading standard Eclipse projects 8
- user role 89

V

- VM startup arguments 83

W

- Web application projects 77
- Web application server
 - configuring launch settings 16
- Web application templates 13
 - adding to an existing project 14
 - using 13
- Web applications
 - running 17
 - web authentication 86
- web authentication 86
- web domain user
 - adding 86
- Web Tools Platform (WTP) 4
- Web user role 89
- webAF projects
 - importing 6, 7, 8
- web.xml 86

X

- XML filters 79

Your Turn

If you have comments or suggestions about *SAS® AppDev Studio™ 3.2 Eclipse Plug-ins: User's Guide*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
E-mail: **yourturn@sas.com**

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
E-mail: **suggest@sas.com**

SAS® Publishing gives you the tools to flourish in any environment with SAS!

Whether you are new to the workforce 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.

SAS® Press Series

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

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: 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, and hard-copy books.

support.sas.com/publishing

SAS® Learning Edition 4.1

Get a workplace advantage, perform analytics in less time, and prepare for the SAS Base Programming exam and SAS Advanced Programming exam with SAS® Learning Edition 4.1. This inexpensive, intuitive personal learning version of SAS includes Base SAS® 9.1.3, SAS/STAT®, SAS/GRAPH®, SAS/QC®, SAS/ETS®, and SAS® Enterprise Guide® 4.1. Whether you are a professor, student, or business professional, this is a great way to learn SAS.

support.sas.com/LE



**THE
POWER
TO KNOW®**