



THE
POWER
TO KNOW.

SAS[®] AppDev Studio[™] 3.4

Eclipse Plug-ins

Migration Guide



The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2009. *SAS® AppDev Studio™ 3.4 Eclipse Plug-ins: Migration Guide*. Cary, NC: SAS Institute Inc.

SAS® AppDev Studio™ 3.4 Eclipse Plug-ins: Migration Guide

Copyright © 2009, SAS Institute Inc., Cary, NC, USA

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

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

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

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

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

1st electronic book, December 2009

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at support.sas.com/publishing 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

Chapter 1 • SAS® AppDev Studio™ 3.4 Eclipse Plug-ins Migration Guide	1
Introduction	1
Accessibility	3
Migrating WebApp Projects That Will Continue to Use Local SAS Foundation Services	3
Migrating WebApp Projects to Use SAS Remote Services	8
Migrating WebApp Projects to Use the SAS Web Infrastructure Platform	13

Chapter 1

SAS® AppDev Studio™ 3.4 Eclipse Plug-ins Migration Guide

Introduction	1
AppDev Studio, Facets, and the SAS Web Infrastructure Platform	1
Migration Options	2
Accessibility	3
Migrating WebApp Projects That Will Continue to Use Local SAS Foundation Services	3
Introduction	3
Creating a BI Server Profile	4
Creating a Host Web Application Project	4
Create a Local Services Deployment Descriptor	4
Add a Context Listener	5
Add Resources to the New Host Project	6
Evaluate the Added Resources	6
Migrating WebApp Projects to Use SAS Remote Services	8
Introduction	8
Creating a BI Server Profile	8
Create a Host Web Application Project	9
Add a Context Listener	9
Add Resources to the Host Project	10
Evaluate the Added Resources	10
Migrating WebApp Projects to Use the SAS Web Infrastructure Platform	13
Introduction	13
Creating a BI Server Profile	13
Create a Host Web Application Project	13
Create the Application Metadata	14
Define a Web Application Entry Point	14
Add Resources to the Host Project	15
Evaluate the Added Resources	15

Introduction

AppDev Studio, Facets, and the SAS Web Infrastructure Platform

SAS AppDev Studio 3.4 Eclipse Plug-ins creates SAS Web applications only for SAS 9.2. AppDev Studio releases 3.0 to 3.3 created Web applications for SAS 9.1.3.

AppDev Studio 3.4 continues to use facets in SAS Web Application projects to add support for SAS features. Facets are a feature of the [Web Tools Platform](#), and define functionality that can be added to a project. Facets also support versioning, which is useful for supporting maintenance updates of SAS AppDev Studio.

Releases of AppDev Studio before 3.4 supported a single SAS facet named “SAS Web Module with WIK”. The WIK was the SAS Web Infrastructure Kit, a group of Java classes and static Web content that provided access to the SAS 9.1.3 Java Components. A SAS 9.1.3 Web application project was defined by the presence of this facet.

AppDev Studio 3.4 does not support the “SAS Web Module with WIK” facet. Instead, it supports two other SAS facets, the “SAS Java Components” facet and the “SAS Web Infrastructure Platform” facet. The “SAS Java Components” facet is the SAS 9.2 equivalent of the “SAS Web Module with WIK” facet. However, the differences between these two facets make it impractical for AppDev Studio 3.4 to support importing SAS Web application projects from prior releases of AppDev Studio. Accordingly, to migrate a SAS Web application project from a prior release, you must create a new SAS Web application project in AppDev Studio 3.4 and then port the old content.

When migrating a SAS Web application project, you can choose to include the “SAS Web Infrastructure Platform” facet and update the project content to use the SAS Web Infrastructure Platform. The SAS Web Infrastructure Platform is a collection of middle-tier services and applications delivered as part of the SAS Integration Technologies package, which provides basic integration services such as authentication. As such, all Business Intelligence (BI) applications, Data Integration applications, and SAS Solutions have access to the SAS Web Infrastructure Platform. For details about the SAS Web Infrastructure Platform support in SAS Web application development, consult the [SAS AppDev Studio 3.4 Developer's Site](#).

In AppDev Studio 3.4, a standard SAS Web Application project is assumed to contain both SAS facets. Consistent with this, the SAS Web Application Project wizard always adds both facets to a new SAS Web application project. Once added, neither SAS facet can be removed from the project. This means the SAS Web Application Project wizard cannot be used to create a SAS Web application project if you do not want to include the “SAS Web Infrastructure Platform” facet. An alternate process for creating a SAS Web application project with only the “SAS Java Components” facet is provided in [“Migrating WebApp Projects That Will Continue to Use Local SAS Foundation Services”](#) on page 3.

Migration Options

All projects must be updated to AppDev Studio 3.3 before migrating to AppDev Studio 3.4. In addition, because of the magnitude of the changes in SAS 9.2, you must migrate AppDev Studio 3.3 projects by creating a new Web application project in AppDev Studio 3.4 and then moving the old content to the new project. You cannot import SAS 9.1.3 Web Application projects into AppDev Studio 3.4, even if those projects were created with prior versions of AppDev Studio. Migrating a project requires that you decide how the new project will surface content. Choose one of the following options:

1. Continue to use local SAS Foundation Services.

This choice requires the least change to existing code, but some features of the SAS Web Infrastructure Platform will not be available, such as integrated logon. For details about this approach, see [“Migrating WebApp Projects That Will Continue to Use Local SAS Foundation Services”](#) on page 3.

2. Modify the Web application to use SAS Remote Services.

This choice requires more changes to the existing code. Some of the Web application's content must have been derived from SAS Web application templates in prior releases

of AppDev Studio. For each Web application template from a prior release of AppDev Studio, AppDev Studio 3.4 has an equivalent template that uses only SAS Remote Services. For details about this approach see [“Migrating WebApp Projects to Use SAS Remote Services” on page 8](#).

3. Modify the Web application to use the SAS Web Infrastructure Platform.

This choice assumes that local SAS Foundation Services will be changed to SAS Remote Services. Thus, the amount of change to the content is similar to using SAS Remote Services. For each Web application template from a prior release of AppDev Studio, AppDev Studio 3.4 has an equivalent template that supports the SAS Web Infrastructure Platform. For details about this approach see [“Migrating WebApp Projects to Use the SAS Web Infrastructure Platform” on page 13](#).

Accessibility

SAS AppDev Studio 3.4 Eclipse Plug-ins includes accessibility and compatibility features that improve the usability of the product for users with disabilities, with exceptions noted below. These features are related to accessibility standards for electronic information technology that were adopted by the U.S. Government under Section 508 of the U.S. Rehabilitation Act of 1973, as amended.

The exceptions are as follows:

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

If you have questions or concerns about the accessibility of SAS products, send e-mail to accessibility@sas.com.

Migrating WebApp Projects That Will Continue to Use Local SAS Foundation Services

Introduction

SAS AppDev Studio 3.4 supports migrating SAS 9.1.3 Web application projects to SAS 9.2 where local SAS Foundation Services are used. Although this migration path involves the least amount of code change, some features provided by the SAS Web Infrastructure Platform, such as integrated logon, will not be available.

The high-level steps for migrating a project that used Local SAS Foundation Services are as follows:

1. Create a BI Server Profile.

2. Create a host Web Application Project.
3. Create a local services deployment descriptor.
4. Add a context listener to deploy the new local SAS Foundation Services.
5. Add resources to the host project.
6. Evaluate the added resources.

Creating a BI Server Profile

If a SAS BI Server Profile does not already exist for the SAS BI installation that the migrated Web application will be developed against, define one. Use the "Create a SAS BI Server Profile" cheat sheet to guide you through the process.

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

If the "Create a SAS BI Server Profile" cheat sheet has already been run, you must reset it before it can be run again. Reset it by right-clicking the cheat sheet name in the tree and selecting **Restart all tasks**.

Creating a Host Web Application Project

Create a new project to host the migrated code. This new SAS Web Application project should contain only the SAS Java Components facet, which is equivalent to the SAS Web Module with WIK facet used by prior versions of SAS AppDev Studio. Create the new project by following these steps:

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

Create a Local Services Deployment Descriptor

AppDev Studio 3.4 does not provide an ADS Local Services deployment descriptor like prior versions of AppDev Studio. Instead, you must duplicate the descriptor used by a SAS BI Web application by following these steps:

1. Open SAS Management Console and use `<sasadmin>@<saspassword>` to connect to the SAS Metadata Server for the SAS BI installation you are developing against.

2. Select the **Plug-ins** tab.
3. Under the Environment Management folder, expand **Foundation Services Manager**.
4. Right-click **SASWIPServices9.2 Local Services**, and then select **Duplicate Service Deployment**.
5. Enter a name for the service deployment that is appropriate for your Web application, for example “SASAppDevStudio3.4 Local Services”.
6. Expand the tree for the new service deployment, and then expand the **Core** folder.
7. Right-click **Logging Service**, and then select **Properties**.
8. Select the **Service Configuration** tab.
9. Click **Configuration**.
10. Select the **Outputs** tab.
11. From the Outputs list, select **SAS_LS_FILE**.
12. Click **Delete**.
 Alternately, click **Edit** and modify the log filename to be specific to your Web application. Ensure that the path to the log file exists. Initially this path is on the AppDev Studio system that is testing the migrated Web application project, but the path must also exist on the remote system when the application is deployed.
13. Click **OK** to confirm all actions and exit the SAS Management Console.

Add a Context Listener

The next step in the migration is to add a context listener that deploys the new local SAS Foundation Services. To add the correct context listener, you need to know whether the original SAS 9.1.3 Web application deployed SAS Remote Services in addition to the local services. To add the context listener, follow these steps:

1. From Eclipse, select **File** ⇒ **New** ⇒ **Other**.
2. Expand the **SAS AppDev Studio** folder.
3. Select **Add Template Content to Project**, and then click **Next**.
4. For the **Project**, select the host project that was created earlier.
5. Expand the **SAS Java Web Application** and then **SAS Foundation Services Support** folders.
6. If the Web application that you are migrating deploys only local SAS Foundation Services, select **Context Listener For Local Services**. If the Web application also deploys SAS Remote Services, select **Context Listener for Remote and Local Services**.
7. Click **Next**.
8. Click **Next** to accept the default Template Configuration Parameters.
9. Select the BI Server Profile for the BI installation whose SAS Metadata Server holds the new local services deployment descriptor (created in the previous section).
10. Click **Advanced**.
11. In the Local Services Deployment section, select **Other**.
12. Click **Browse** and select the service deployment descriptor that you created earlier.

13. Click **OK**.

If the SAS Metadata Server for the BI installation is running, you can click **Test Configuration** to verify that the service deployment or deployments can be read from the metadata server.

14. Click **Finish**.

The host SAS 9.2 Web Application project is complete. You can now add resources from the old SAS 9.1.3 Web application to this new host project.

Add Resources to the New Host Project

Add resources from the old SAS 9.1.3 Web application to the new host project. Note that the static content from the SAS 9.1.3 Web application does not need to be copied because the SAS 9.2 version of this static content already exists in the new host project. Only resources added to the original SAS 9.1.3 Web application project need to be migrated.

Copy files to the new host project by dragging or copying them from the file system and then dropping or pasting them into the Project Explorer, Package Explorer, or Navigator views. If you copy files to your project directory using the file system, select **Refresh** in the Eclipse context menu to make the copied files visible in the workspace. You can also use the Eclipse Import wizard by following these steps:

1. From Eclipse, select **File** ⇒ **Import**.
2. Expand the **General** folder.
3. Select **File System**.
4. Click **Next**.
5. Specify the source files to import. Do not overwrite any existing files in the new host project.

Evaluate the Added Resources

Introduction

Each migrated resource must be evaluated for changes that are needed to upgrade it to SAS 9.2. The following sections describe the typical resources and the necessary changes.

Migrating SBIP URLs

Review all appropriate imports for the presence of SAS Business Intelligence Platform (SBIP) URLs and update them to the new SAS 9.2 format. In SAS 9.1.3, an SBIP URL typically began with **SBIP://Foundation/**, in SAS 9.2, they begin with **SBIP://METASERVER/**.

The BIP Tree used in SAS 9.1.3 does not exist in SAS 9.2, and if you recreated it in SAS 9.2 and altered the path to it when you imported the metadata objects, then you must specify the new path in SBIP URLs. If you only need a starting folder, use **SBIP://METASERVER/Shared Data(Folder)**.

Migrating JSP Pages

Change style sheet references from the old style:

```
<link href="styles/sasComponents.css" rel="STYLESHEET" type="text/css">
```

to the new reference:

```
<sas:InitializeComponents />
```

All SAS AppDev Studio 3.3 Web application viewer JSPs use the old reference style.

If your Web application might one day be upgraded to use the SAS Web Infrastructure Platform and its support for themes, save time now and change the CSS reference to the following:

```
<% if (request.getAttribute(CommonKeys.DISPLAY_THEME_OBJECT) != null){ %>
<sas:StyleSheet />
<% / %>
<sas:InitializeComponents />
```

If you change the CSS reference, also add the import:

```
<%@ page import="com.sas.web.keys.CommonKeys" %>
```

Viewer JSPs from SAS Web application templates should not need additional changes. Modified and custom JSPs should be reviewed to determine that the APIs and SAS tags used remain compatible in SAS 9.2. The current SAS Custom Tag Reference can be found at <http://support.sas.com/rnd/gendoc/bi92/api/Components/taglibs/sas/overview.html>.

Migrating Java Files

Review all code for deprecated APIs. Review JDBC code for the correct host and port information in URLs to SAS 9.2 BI servers.

If your SAS 9.1.3 Web application servlet used `ExamplesSessionBindingListener`, you can either copy the old Java file from your SAS 9.1.3 Web application to the new project, or you can copy the Java file from a SAS 9.2 Web application project that has had at least one template added to it.

Because SAS 9.2 uses a version of Java that supports generics, code from SAS 9.1.3 will generate warnings about raw types. You can either ignore the warnings or update the code to use generics.

Migrating Other Resources

Migrating non-JSP and non-Java resources includes, at minimum, copying to the new project `web.xml` declarations from the SAS 9.1.3 Web application that do not exist in new project's `web.xml`.

For servlets, copy the `<servlet>` and `<servlet-mapping>` declarations and any other related declarations from the old `web.xml`. Also review the initialization parameters for all servlet declarations. For example, the 9.1.3 Web Application templates stored the user credentials in the initialization parameters, and they might need updating or removal.

For copied Report Viewer Servlet templates, update the URL specified in the "viewer" initialization parameter. In SAS 9.1.3, this URL followed the format `http://<host>:<port>/SASWebReportViewer/logonFromPortal.do`. Update the `<host>` and `<port>` to match the middle-tier host and port of your SAS 9.2 BI installation. If your SAS 9.2 BI installation included SAS Web Report Studio rather than SAS Web Report Viewer, replace "SASWebReportViewer" with "SASWebReportStudio". Lastly, change the resource referenced from "logonFromPortal.do" to "logonAndRender.do".

In AppDev Studio 3.3, because a user with a valid `_SESSIONID` could access SAS Web applications without authenticating as long as the session remained active, developers included the `UrlReplayBlocker` filter. (See [SAS Note 20591](#).) Accordingly, if the Web application being migrated displays URLs that expose session IDs, copy the `UrlReplayBlocker` filter declaration and mapping to the new Web application. In SAS 9.2,

SAS Web application URLs that are part of a BI installation are not vulnerable to this issue and do not need this filter.

Add any needed custom jars to the project. For classes that are available in jars found in the SAS Versioned Jar Repository, right-click on the Java file in the Project Explorer or Package Explorer view and select **Organize SAS Imports**. If the Java file is open in a Java editor, right-click in the editor and select **Source** ⇒ **Organize SAS Imports**. You can also add missing classes individually using the Quick Fix that is marked with a SAS icon. If the SAS 9.1.3 Web application had custom jars in `/WEB-INF/lib/` that are not already present or not available in the SAS Versioned Jar Repository, copy those jars to the new project's `/WEB-INF/lib/`.

You might need to recreate dependencies for jars specified in the Java EE Module Dependencies list. Also review the versions of those jars to determine whether you need to use a newer version with the jars already in the SAS 9.2 Web application project.

Because running the SAS 9.2 middle-tier Web server with a Java security manager is not recommended, AppDev Studio 3.4 does not provide Java policy files for SAS Web application projects.

Migrating WebApp Projects to Use SAS Remote Services

Introduction

Migrating SAS AppDev Studio 3.3 Web application projects to SAS 9.2 where SAS Remote Services are used is supported by SAS AppDev Studio 3.4. Content in a SAS 9.1.3 Web application that was created using an AppDev Studio template can be migrated by creating the equivalent project in AppDev Studio 3.4 and then porting any customizations. Although this migration path avoids the need for a local services deployment descriptor, some features provided by the SAS Web Infrastructure Platform, such as integrated logon, will not be available.

The high-level steps for migrating a project that used SAS Remote Services are as follows:

1. Create a BI Server Profile.
2. Create a host Web application Project.
3. Add a context listener to deploy the new local SAS Foundation Services.
4. Add resources to the host project.
5. Evaluate the added resources.

Creating a BI Server Profile

If a SAS BI Server Profile does not already exist for the SAS BI installation that the migrated Web application will be developed against, define one. Use the "Create a SAS BI Server Profile" cheat sheet to guide you through the process.

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

If the "Create a SAS BI Server Profile" cheat sheet has already been run, you must reset it before it can be run again. Reset it by right-clicking the cheat sheet name in the tree and selecting **Restart all tasks**.

Create a Host Web Application Project

Create a new project to host the migrated code. This new SAS Web Application project should contain only the SAS Java Components facet, which is equivalent to the SAS Web Module with WIK facet used by prior versions of SAS AppDev Studio. Create the new project by following these steps:

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

Add a Context Listener

The next step in the migration is to add a context listener that deploys the SAS Remote Services. To add the context listener, follow these steps:

1. From Eclipse, select **File** ⇒ **New** ⇒ **Other**.
2. Expand the **SAS AppDev Studio** folder.
3. Select **Add Template Content to Project**, and then click **Next**.
4. For the **Project**, select the host project that was created earlier.
5. Expand the **SAS Java Web Application** and then **SAS Foundation Services Support** folders.
6. Select **Context Listener for Remote Services**.
7. Click **Next**.
8. Click **Next** to accept the default Template Configuration Parameters.
9. Select the BI Server Profile for the BI installation whose SAS Metadata Server holds the new local services deployment descriptor (created in the previous section).
10. If the SAS Metadata Server for that BI installation is running, you can click **Test Configuration** to verify the remote service deployment can be read from the metadata server.
11. Click **Finish**.

The host SAS 9.2 Web Application project is complete. You can now add resources from the old SAS 9.1.3 Web application to this new host project.

Add Resources to the Host Project

Add resources from the old SAS 9.1.3 Web application to the new host project. Note that the static content from the SAS 9.1.3 Web application does not need to be copied because the SAS 9.2 version of this static content already exists in the new host project. Only resources added to the original SAS 9.1.3 Web application project need to be migrated.

Copy files to the new host project by dragging or copying them from the file system and then dropping or pasting them into the Project Explorer, Package Explorer, or Navigator views. If you copy files to your project directory using the file system, select **Refresh** in the Eclipse context menu to make the copied files visible in the workspace. You can also use the Eclipse Import wizard by following these steps:

1. From Eclipse, select **File** ⇒ **Import**.
2. Expand the **General** folder.
3. Select **File System**.
4. Click **Next**.
5. Specify the source files to import. Do not overwrite any existing files in the new host project.

Evaluate the Added Resources

Introduction

Each migrated resource must be evaluated for changes that are needed to upgrade it to SAS 9.2. The following sections describe the typical resources and the necessary changes.

Migrating SBIP URLs

Review all appropriate imports for the presence of SAS Business Intelligence Platform (SBIP) URLs and update them to the new SAS 9.2 format. In SAS 9.1.3, an SBIP URL typically began with **SBIP://Foundation/**, in SAS 9.2, they begin with **SBIP://METASERVER/**.

The BIP Tree used in SAS 9.1.3 does not exist in SAS 9.2, and if you recreated it in SAS 9.2 and altered the path to it when you imported the metadata objects, then you must specify the new path in SBIP URLs. If you only need a starting folder, use **SBIP://METASERVER/Shared Data(Folder)**.

Migrating JSP Pages

When you create a SAS Web application, a viewer JSP is provided. This JSP does not require any changes unless there were customizations to the SAS 9.1.3 Web application.

Change style sheet references from the old style:

```
<link href="styles/sasComponents.css" rel="STYLESHEET" type="text/css">
```

to the new reference:

```
<sas:InitializeComponents />
```

All SAS AppDev Studio 3.3 Web application viewer JSPs use the old reference style.

If your Web application might one day be upgraded to use the SAS Web Infrastructure Platform and its support for themes, save time now and change the CSS reference to the following:

```
<% if (request.getAttribute(CommonKeys.DISPLAY_THEME_OBJECT) != null){ %>
<:sas:StyleSheet />
<% / %>
<:sas:InitializeComponents />
```

If you change the CSS reference, also add the import:

```
<@ page import="com.sas.web.keys.CommonKeys" %>
```

Modified and custom JSPs should be reviewed to determine that the APIs and SAS tags used remain compatible in SAS 9.2. The current SAS Custom Tag Reference can be found at <http://support.sas.com/rnd/gendoc/bi92/api/Components/taglibs/sas/overview.html>.

Migrating Java Files

Review all code for deprecated APIs. Because this project type uses only SAS Remote Services, modify any Java code that uses local services so that it uses remote services.

Also, because only remote services are deployed, code using the `DiscoveryService` to obtain services now returns remote services rather than local services. For example, code that creates SAS `UserContexts` and SAS `SessionContexts`. Although it might seem that the code that uses these remote services does not need altering, you do need to modify it to avoid side effects that can occur when you switch from local to remote services. For example, both the `Information Map Viewer Servlet` and `Report Viewer Servlet` templates create an `InformationServicesSelector` object with code like the following:

```
List repos = sessionContext.getUserContext().getRepositories();
RepositoryInterface reposInt = (RepositoryInterface) repos.get(0);

sas_informationSelector = new InformationServicesSelector(reposInt,
    DEFAULT_REPOSITORY_FOLDER, null);
```

When using this code with a SAS 9.2 BI installation and remote services, the `getRepositories()` call returns a list with two items. The second item in the list should be used as the `RepositoryInterface` (the one whose `RepositoryInterface.getUrl()` returns a string that starts with "omi:").

In SAS AppDev Studio 3.4, the templates do not use the `RepositoryInterface`. Instead, they use a different constructor for `InformationServicesSelector`, as shown in the following code:

```
sas_informationSelector = new InformationServicesSelector(userContext,
    true, <location URL>, null);
```

The `<location URL>` is the initial folder displayed by the `InformationServicesSelector` and can be set to the `DEFAULT_REPOSITORY_FOLDER` constant. In the case of the SAS AppDev Studio 3.4 templates, the `<location URL>` is set to the user's default folder, which is built using code similar to the following:

```
String user = userContext.getName();
String locationURL = "SBIP://METASERVER/Users/" + user + "(Folder)";
```

Although you can keep the AppDev Studio 3.3 `getRepositories` code, provided that you fix it to use the "omi" repository (the second item in the returned list), the new constructor for `InformationServicesSelector` is a better solution and should replace instances of the old code.

In the case of servlets, you should recreate old SAS Web application servlets using the equivalent template in SAS AppDev Studio 3.4. Doing so creates the associated servlet with code that uses SAS Remote Services and other code improvements, like the use of `InformationServicesSelector` mentioned above. When templates in AppDev Studio 3.4 ask you to provide settings like host names and port numbers, you can enter placeholder values that can be changed when the new template is in place and customizations are migrated from the 9.1.3 version of the template.

Most servlets in the SAS 9.1.3 Web application templates used the `ExamplesSessionBindingListener`. You should use the new version of this object from an AppDev Studio 3.4 template rather than copy one from an old application. The new object supports the "SAS Stored Process Servlet (uses SAS FS)" template, which can be added to this project.

Because SAS 9.2 uses a version of Java that supports generics, code from 9.1.3 generates warnings about the use of raw types. Code from recreated templates already uses generics.

Migrating Other Resources

Migrating non-JSP and non-Java resources includes, at minimum, copying to the new project `web.xml` declarations from the SAS 9.1.3 Web application that do not exist in the new project's `web.xml`.

For servlets, copy the `<servlet>` and `<servlet-mapping>` declarations and any other related declarations from the old `web.xml`. Also review the initialization parameters for all servlet declarations. For example, the 9.1.3 Web Application templates stored the user credentials in the initialization parameters, and they might need updating or removal.

For copied Report Viewer Servlet templates, update the URL specified in the "viewer" initialization parameter. In SAS 9.1.3, this URL followed the format `http://<host>:<port>/SASWebReportViewer/logonFromPortal.do`. Update the `<host>` and `<port>` to match the middle-tier host and port of your SAS 9.2 BI installation. If your SAS 9.2 BI installation included SAS Web Report Studio rather than SAS Web Report Viewer, replace "SASWebReportViewer" with "SASWebReportStudio". Lastly, change the resource referenced from "logonFromPortal.do" to "logonAndRender.do".

In AppDev Studio 3.3, because a user with a valid `_SESSIONID` could access SAS Web applications without authenticating as long as the session remained active, developers included the `UrlReplayBlocker` filter. (See [SAS Note 20591](#).) If the Web application being migrated displays URLs that are vulnerable to this issue, copy the `UrlReplayBlocker` filter declaration and mapping to the new Web application. In SAS 9.2, SAS Web application URLs that are part of the BI installation are not vulnerable to this issue and do not need this filter.

Add any needed custom jars to the project. For classes that are available in jars found in the SAS Versioned Jar Repository, right-click on the Java file in the Project Explorer or Package Explorer view and select **Organize SAS Imports**. If the Java file is open in a Java editor, right-click in the editor and select **Source** ⇒ **Organize SAS Imports**. You can also add missing classes individually using the Quick Fix that is marked with a SAS icon. If the SAS 9.1.3 Web application had custom jars in `/WEB-INF/lib/` that are not already present or not available in the SAS Versioned Jar Repository, copy those jars to the new project's `/WEB-INF/lib/`.

You might need to recreate dependencies on jars specified in the Java EE Module Dependencies. Review the versions of those jars to determine whether you need to use a newer version with the jars already in the SAS 9.2 Web application project.

Because running the SAS 9.2 middle-tier Web server with a Java security manager is not recommended, AppDev Studio 3.4 does not provide Java policy files for SAS Web application projects.

Migrating WebApp Projects to Use the SAS Web Infrastructure Platform

Introduction

Migrating SAS AppDev Studio 3.3 Web Application projects to SAS 9.2 where the SAS Web Infrastructure Platform is used provides access to the features of the SAS Web Infrastructure Platform, which many of the SAS AppDev Studio 3.4 templates use. Content in a SAS 9.1.3 Web application that was created using an AppDev Studio template can be migrated by creating a Web application project for Web Infrastructure Platform content using AppDev Studio 3.4 and then porting any customizations.

The high-level steps for migrating a project to use the SAS Web Infrastructure Platform are as follows:

1. Create a BI Server Profile.
2. Create a host Web Application Project.
3. Create the application metadata.
4. Define a Web application entry point.
5. Add resources to the host project.
6. Evaluate the added resources.

Creating a BI Server Profile

If a SAS BI Server Profile does not already exist for the SAS BI installation that the migrated Web application will be developed against, define one. Use the "Create a SAS BI Server Profile" cheat sheet to guide you through the process.

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

If the "Create a SAS BI Server Profile" cheat sheet has already been run, you must reset it before it can be run again. Reset it by right-clicking the cheat sheet name in the tree and selecting **Restart all tasks**.

Create a Host Web Application Project

Create a new SAS Web Application project to host the migrated code by following these steps:

1. From Eclipse, select **File** ⇒ **New** ⇒ **Project**.
2. Expand the **SAS AppDev Studio** folder.
3. Select **SAS Web Application Project**, and click **Next**.
4. Specify a project name and click **Next**.

This name is used as the context name. The name must not contain spaces.

5. Select the **Add Template Content** check box.
6. Expand the **SAS Java Web Application** and then **SAS Web Infrastructure Platform Support** folders.
7. Select **SAS Web Infrastructure Platform Application Metadata Creation**, and click **Next**.
8. Select the BI Server Profile that matches the BI installation that you plan to develop against.
9. Enter the **Application Name** and a **Description**.
The name can contain spaces.
10. Clear the **Application ID** field.
The Application ID should be blank unless there is a specific need to find the application metadata using an ID.
11. Specify the **Port** for the test server, and then click **Finish**.

Although the SAS Web Application Project needed for the migration has now been created, you must still create the needed metadata.

Create the Application Metadata

Because you added to the project a SAS Web Infrastructure Platform Application Metadata Creation template, the project contains a metadata folder. Create the necessary metadata for the project by following these steps:

1. Ensure that the SAS Metadata Server is running in the BI installation whose SAS BI Server Profile was selected earlier.
2. From Eclipse, select **Window** ⇒ **Show View** ⇒ **Console**.
3. From Eclipse, select **Run** ⇒ **External Tools** ⇒ **External Tools Configurations**.
4. Expand the **Ant Build** folder
5. Select the Ant configuration **<project name> Create Metadata**, where **<project name>** is the host project created earlier.
6. Click **Run**.

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

Define a Web Application Entry Point

The entry point to the application is where users are directed following a successful logon. The default entry point, specified in the metadata created earlier, is the root of the Web application. A welcome file, as defined by the servlet, needs to exist at the root location to serve as the entry point.

You can add your own welcome file, or you can add a welcome file using the “Example Welcome Page” template. The template welcome file displays a link for each SAS Web application template added to the Web application so that they can be easily executed. Add a template welcome file by following these steps:

1. From Eclipse, select **File** ⇒ **New** ⇒ **Other**.
2. Expand **SAS AppDev Studio**.

3. Select **Add Template Content to Project**, and click **Next**.
4. Expand **SAS Java Web Application and SAS Web Infrastructure Platform Support**.
5. Select **Examples Welcome Page**, and click **Next**.
6. Edit the name of the Welcome file and then click **Finish**.

The Welcome file is created. You can now add other SAS templates to the project and execute them from the Welcome file.

Add Resources to the Host Project

Add resources from the old SAS 9.1.3 Web application to the new host project. Note that the static content from the SAS 9.1.3 Web application does not need to be copied because the SAS 9.2 version of this static content already exists in the new host project. Only resources added to the original SAS 9.1.3 Web application project need to be migrated.

Copy files to the new host project by dragging or copying them from the file system and then dropping or pasting them into the Project Explorer, Package Explorer, or Navigator views. If you copy files to your project directory using the file system, select **Refresh** in the Eclipse context menu to make the copied files visible in the workspace. You can also use the Eclipse Import wizard by following these steps:

1. From Eclipse, select **File** ⇒ **Import**.
2. Expand the **General** folder.
3. Select **File System**.
4. Click **Next**.
5. Specify the source files to import. Do not overwrite any existing files in the new host project.

Evaluate the Added Resources

Introduction

Each migrated resource must be evaluated for changes that are needed to upgrade it to SAS 9.2. The following sections describe the typical resources and the necessary changes.

Migrating SBIP URLs

Review all appropriate imports for the presence of SAS Business Intelligence Platform (SBIP) URLs and update them to the new SAS 9.2 format. In SAS 9.1.3, an SBIP URL typically began with **SBIP://Foundation/**, in SAS 9.2, they begin with **SBIP://METASERVER/**.

The BIP Tree used in SAS 9.1.3 does not exist in SAS 9.2, and if you recreated it in SAS 9.2 and altered the path to it when you imported the metadata objects, then you must specify the new path in SBIP URLs. If you only need a starting folder, use **SBIP://METASERVER/Shared Data(Folder)**.

Migrating JSP Pages

When you create a SAS Web application, a viewer JSP is provided. This JSP does not require any changes unless there were customizations to the SAS 9.1.3 Web application.

Change style sheet references from the old style:

```
<link href="styles/sasComponents.css" rel="STYLESHEET" type="text/css">
```

to the following:

```
<% if (request.getAttribute(CommonKeys.DISPLAY_THEME_OBJECT) != null){ %>
<:StyleSheet />
<% } %>
<:InitializeComponents />
```

All SAS AppDev Studio 3.3 Web application viewer JSPs use the old reference style.

Also add the import:

```
<%@ page import="com.sas.web.keys.CommonKeys" %>
```

Migrating Java Files

Review all code for deprecated APIs. Modify Java code where necessary to use the features of the SAS Web Infrastructure, and to use SAS Remote Services instead of local SAS Foundation Services. These changes might involve the removal or simplification of code.

For example, because the SAS Web Infrastructure Platform configuration in the SAS 9.2 Web application deploys "Remote Services" for you, you do not need to copy the `FoundationServicesContextListener` from the SAS 9.1.3 Web application. Also unneeded are the associated properties files in the SAS 9.1.3 Web application's `/WEB-INF/conf/` directory.

The main benefit of using the SAS Web Infrastructure Platform integrated logon feature is that a remote SAS `UserContext` and remote SAS `SessionContext` for the logged on user are automatically placed in the HTTP session and do not need to be created. They are retrieved using the session attributes defined by the `REMOTE_USER_CONTEXT` and `REMOTE_SESSION_CONTEXT` constants, which are defined in the `com.sas.web.keys.CommonKeys` class. Consequently, code from a SAS 9.1.3 Web application that creates a SAS `UserContext` or SAS `SessionContext` can be replaced with one of the following statements:

```
UserContextInterface userContext =
    (UserContextInterface)session.getAttribute(CommonKeys.REMOTE_USER_CONTEXT);

SessionContextInterface remoteSessionContext =
    (SessionContextInterface)session.getAttribute(CommonKeys.REMOTE_SESSION_CONTEXT);
```

Also, because only remote services are deployed, code using the `DiscoveryService` to obtain services now returns remote services rather than local services. For example, code that creates SAS `UserContexts` and SAS `SessionContexts`. Although it might seem that the code that uses these remote services does not need altering, you do need to modify it to avoid side effects that can occur when you switch from local to remote services. For example, both the `Information Map Viewer Servlet` and `Report Viewer Servlet` templates create an `InformationServicesSelector` object with code like the following:

```
List repos = sessionContext.getUserContext().getRepositories();
RepositoryInterface reposInt = (RepositoryInterface)repos.get(0);

sas_informationSelector = new InformationServicesSelector(reposInt,
    DEFAULT_REPOSITORY_FOLDER, null);
```

When using this code with a SAS 9.2 BI installation and remote services, the `getRepositories()` call returns a list with two items. The second item in the list should be used as the `RepositoryInterface` (the one whose `RepositoryInterface.getUrl()` returns a string that starts with "omi:").

In SAS AppDev Studio 3.4, the templates do not use the `RepositoryInterface`. Instead, they use a different constructor of `InformationServicesSelector`. The following code can be used as a replacement for the code above:

```

sas_informationSelector = new InformationServicesSelector(userContext,
    true, <location URL>, null)

```

The `<location URL>` is the initial folder displayed by the `InformationServicesSelector` and can be set to the `DEFAULT_REPOSITORY_FOLDER` constant. In the case of the SAS AppDev Studio 3.4 templates, the `<location URL>` is set to the user's default folder, which is built using code similar to the following:

```

String user = userContext.getName();
String locationURL = "SBIP://METASERVER/Users/" + user + "(Folder)";

```

In the case of servlets, you should recreate old SAS Web application servlets using the equivalent template in SAS AppDev Studio 3.4. Doing so creates the associated servlet with code that makes use of the SAS Web Infrastructure Platform and other code improvements, like the use of `InformationServicesSelector` mentioned above. When templates in AppDev Studio 3.4 ask you to provide settings like host names and port numbers, you can enter placeholder values that can be changed when the new template is in place and customizations are migrated from the 9.1.3 version of the template.

Most servlets in the SAS 9.1.3 Web application templates used the `ExamplesSessionBindingListener`. You should use the new version of this object from an AppDev Studio 3.4 template rather than copy one from an old application. The new object supports the "SAS Stored Process Servlet (uses SAS FS)" template, which can be added to this project.

Because SAS 9.2 uses a version of Java that supports generics, code from 9.1.3 generates warnings about the use of raw types. Code from recreated templates already uses generics.

Migrating Other Resources

Migrating non-JSP and non-Java resources includes, at minimum, copying to the new project `web.xml` declarations from the SAS 9.1.3 Web application that do not exist in the new project's `web.xml`.

For servlets, copy the `<servlet>` and `<servlet-mapping>` declarations and any other related declarations from the old `web.xml`. Also review the initialization parameters for all servlet declarations. For example, the 9.1.3 Web Application templates stored user credentials in the initialization parameters, and they might need updating or removal.

For copied Report Viewer Servlet templates, update the URL specified in the "viewer" initialization parameter. In SAS 9.1.3, this URL followed the format `http://<host>:<port>/SASWebReportViewer/logonFromPortal.do`. Replace that URL with `Director?_directive=BIViewReport`, which uses the SAS Web Infrastructure Platform Directive Service via the "director" servlet. The Directive Service enables SAS Web applications to specify URLs in their metadata, which other SAS Web applications can retrieve via the directive name. This avoids including in the URL information that is specific to a particular SAS BI installation, and eliminates the need to modify the URL when moving the Web application to a different SAS BI installation.

In AppDev Studio 3.3, because a user with a valid `_SESSIONID` could access SAS Web applications without authenticating as long as the session remained active, developers included the `UrlReplayBlocker` filter. (See [SAS Note 20591](#).) Because the SAS 9.2 Web application is using the integrated logon, it is not vulnerable to this issue, and the `UrlReplayBlocker` does not need to be migrated.

Add any needed custom jars to the project. For classes that are available in jars found in the SAS Versioned Jar Repository, right-click on the Java file in the Project Explorer or

Package Explorer view and select **Organize SAS Imports**. If the Java file is open in a Java editor, right-click in the editor and select **Source** ⇒ **Organize SAS Imports**. You can also add missing classes individually using the Quick Fix that is marked with a SAS icon. If the SAS 9.1.3 Web application had custom jars in `/WEB-INF/lib/` that are not already present or not available in the SAS Versioned Jar Repository, copy those jars to the new project's `/WEB-INF/lib/`.

You might need to recreate dependencies on jars specified in the Java EE Module Dependencies. Also review the versions of those jars to determine whether you need to use a newer version with the jars already in the SAS 9.2 Web application project.

Because running the SAS 9.2 middle-tier Web server with a Java security manager is not recommended, AppDev Studio 3.4 does not provide Java policy files for SAS Web application projects.

Your Turn

We welcome your feedback.

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