SAS® AppDev Studio™ 3.2

Migration Guide

THE POWER TO KNOW®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2006. *SAS® AppDev™ Studio 3.2: Migration Guide*. Cary, NC: SAS Institute Inc.

SAS® AppDev Studio™ 3.2: Migration Guide

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

1st printing, July 2006

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.

Table of Contents

Introduction

The SAS AppDev Studio 3.2 Eclipse Plug-ins provide enhanced SAS application development support for developers using the open source Eclipse IDE. For more information about Eclipse software, see www.eclipse.org.

Many of the features of webAF software have been ported to Eclipse and delivered in this set of plug-ins, along with new capabilities and enhancements that interoperate with new features in the SAS®9 Business Intelligence Platform.

The purpose of this document is to provide users of SAS AppDev Studio 3.0[1] with guidance on upgrading to this new version of the product. The document includes updated system requirements for the product, information on how to migrate existing development projects forward, guidelines for getting the most from the new features, and a list of obsolete or significantly changed functionality.

New Features

SAS AppDev Studio 3.2 contains many new features. The primary change is that the webAF development tools are now provided in the SAS AppDev Studio Eclipse Plug-ins, which are a set of plug-ins for the open-source Eclipse IDE.

The SAS AppDev Studio Eclipse Plug-ins provide extensions for Eclipse to support the development of SAS applications. Two new project types that are fundamental to this support are the SAS Java Project and the SAS Web Application Project. There is also the new associated SAS Versioned Jar Repository, which is a collection of reusable SAS library JAR files and their dependencies. When you create a SAS project, a SAS Classpath Container provides access to the SAS Versioned Jar Repository. The SAS Classpath Container manages the underlying SAS library JAR file dependencies for your project. So, as you add features to your application that rely on certain SAS library JAR file features, you can easily manage the resulting dependencies for your project. In fact, much of this effort is performed for you automatically by the SAS AppDev Studio Eclipse Plug-ins. In addition, the plug-ins provide templates that support the development of a variety of applications and Web applications, including SAS Information Delivery Portal Portlets, SAS Management Console and Data Integration Studio Plug-ins, Web-based reports, and OLAP solutions.

Users of prior versions of SAS AppDev Studio can use the import features in the plug-ins to import and migrate existing SAS AppDev Studio 3.0 projects into Eclipse. The plug-ins also provide an export facility to create Web Application Archive (WAR) files or Java Archive (JAR) files from your project content.

Finally, the plug-ins enhance the standard Eclipse Java editor to provide assistance when you create applications based on SAS. The plug-ins enable the integration of the SAS 9.1 BI Java Documentation with the Eclipse help system. The SAS Organize Imports action and “Import from Repository” Quick Fix feature automatically manage project dependencies. Code snippets support commonly performed development tasks such as deploying SAS Foundation Services or connecting to a SAS Metadata Server or a

[1] Note that wherever this document refers to SAS AppDev Studio 3.0, it also refers to any of the SAS AppDev Studio 3.0 updates, as well, unless specifically stated otherwise.

Workspace Server, and a Quick Assist operation supports model/view connections between common Java viewer widgets and the back-end data models.

Changes to the SAS AppDev Studio System Requirements

Supported Versions of SAS Software

SAS AppDev Studio 3.2 supports SAS 9.1.3 Service Pack 4 software as well as SAS Release 8.2. This is the last planned release that will support the SAS 8.2 servers. Users should update their applications developed with SAS AppDev Studio 3.2 to work with SAS 9.1.3 Service Pack 4 software to enable migration forward to future releases of SAS AppDev Studio software.

Java Platform Requirements

The SAS AppDev Studio 3.2 requires Java 2 Standard Edition (J2SE) 1.4.2_09 or higher for both application development and execution of applications or applets at run time. For execution of Web applications, SAS AppDev Studio 3.2 supports the Web browsers, application servers, and Java run times as defined in the *Third Party Software for SAS 9.1.3 Foundation with Service Pack 4* (support.sas.com/documentation/configuration/thirdpartysupport/v913sp4/thirdparty913sp4.html) document.

Eclipse Platform Requirements

The SAS AppDev Studio 3.2 Eclipse Plug-ins require the Eclipse 3.1.2 Software Development Kit (SDK) and Eclipse Web Tools Platform (WTP) 1.0.1, or later WTP 1.0.x maintenance release. These files can be downloaded from the Eclipse Foundation (download.eclipse.org) and installed manually or the SAS AppDev Studio 3.2 Eclipse Plug-ins installer can install them automatically for you.

See the *AppDev Studio Java Components 3.2 System Requirements* document for complete system requirements information. A hard copy of the document is provided in the product packaging. An online version is available from the SAS AppDev Studio Developer's Site (support.sas.com/rnd/appdev).

Changes to the SAS AppDev Studio Product Bundle

As noted above, one of the most significant changes to the SAS AppDev Studio product bundle is the port of webAF software to an Eclipse-based implementation. This enables our users to benefit from the industry-leading Eclipse development tools and third-party plug-ins, while also taking advantage of the SAS development capabilities provided by the SAS AppDev Studio Eclipse Plug-ins.

This release of SAS AppDev Studio moves away from some older technology. The following parts of SAS AppDev Studio bundle now have Level C support from the Technical Support staff at SAS. For more information about levels of support, see the Technical Support Web site (support.sas.com/techsup/support.html#non_current).

- **SAS AppDev Studio Middleware Server (MWS)** is no longer provided. As indicated in the *SAS AppDev Studio 3.0 Migration Guide*, the functionality provided by the MWS has been replaced by the scalability features provided by SAS Integration Technologies (support.sas.com/rnd/appdev/itech/index.htm). For more information about servers, see "SAS Server Connections and Scalability Features" at the end of this document.

- **webEIS report builder** is no longer included in the product bundle. webEIS users should migrate to the newer SAS reporting products such as SAS Web Report Studio or the SAS Web OLAP Viewer for Java.

- **SAS AppDev Studio Server-Side Catalogs**, which support connections to certain SCL-based data objects on the SAS server, are included but have been deprecated in favor of using more standard and scalable data connection technologies such as JDBC (for relational data) and the SAS®9 Java OLAP interfaces (for OLAP data).

  Note: This is the last release of SAS AppDev Studio that will include these catalogs.

Build Environment Changes

In SAS AppDev Studio 3.0, webAF software generated Apache Ant build scripts to support the individual project builds. For more information about Apache Ant, see ant.apache.org. Provision was made for customizing the Ant scripts through the use of custom build tasks. Eclipse allows the use of Ant, but the SAS AppDev Studio 3.2 Eclipse Plug-ins do not automatically use this feature. For more information about using Ant with Eclipse, see the Eclipse Workbench User's Guide (help.eclipse.org/help31/index.jsp?topic=/org.eclipse.platform.doc.user/gettingStarted/qs-93_project_builder.htm).

One of the differences between webAF 3.0 software and the SAS AppDev Studio 3.2 Eclipse Plug-ins involves how Eclipse manages *projects* within a containing *workspace*. In webAF software, only one project was open at a time. There was no provision for sharing classes from one project to the next. The Eclipse IDE provides the notion of a workspace, which is a virtual concept that enables you to have multiple projects open at the same time. With multiple projects open, it is possible to share the classes between the projects. In fact, this is the way projects and libraries are routinely constructed in Eclipse. While the workspace definition is contained in a physical location, the projects that it encompasses do not have to live at that same physical location. However, when a webAF 3.0 project is imported into an Eclipse workspace, the project files are copied to the physical location of the workspace. This is different from standard Eclipse behavior, where projects are not copied on import.

Caution: In SAS AppDev Studio 3.0, you had to install using a path that did not include spaces. For the SAS AppDev Studio 3.2 Eclipse Plug-ins, this restriction now applies to the path of your Workspace in order to prevent problems testing Web applications. In the Windows environment, this means overriding the default location, which is the Documents and Settings folder.

Project Conversion Considerations for webAF 3.0 Users

General Project Conversion Considerations

SAS AppDev Studio 3.2 can be installed side-by-side with the previous release. Rather than being updated in place, old projects can be copied out of the SAS AppDev Studio 3.0 project area and into the Eclipse workspace using the webAF Project Import feature of the plug-ins (by selecting **File > Import > webAF Project Import** from the menu bar).

Caution: SAS AppDev Studio 3.2 only supports the migration of projects from SAS AppDev Studio 3.0. It will not import projects from earlier versions.

During the import process, the Java source files that were generated by webAF software are modified to remove artifacts that were required by the webAF 3.0 source editor. These changes are only in code comments and have no impact on the content of the compiled Java class files.

In webAF 3.0 software, it was possible for you to add files to projects without webAF being notified of their existence. For example, you could add files to the project build script directly rather than by using the *Add File to Project* command. In such cases, the *webAF Project Import* feature in the SAS AppDev Studio 3.2 Eclipse Plug-ins may fail to include these files during the import process. Make sure that these files have been explicitly added to your webAF 3.0 project before migrating it to the SAS AppDev Studio 3.2 Eclipse Plug-ins.

SAS AppDev Studio 3.0 included the ability to create applets and applications using a visual drag-and-drop builder. This provided a WYSIWYG experience when designing your interface. This technology is not yet provided in the plug-ins. Visual editing of Java user interfaces within Eclipse is provided by the Eclipse Visual Editor Project or though various commercial add-in products for Eclipse. For more information about the Eclipse Visual Editor Project, see www.eclipse.org/vep.

Applet Project Conversion

In webAF 3.0 software, a background Tomcat server was created to test applet projects in an environment closer to the deployed environment. webAF software also displayed the applet using a standard Web browser. The Eclipse Run as Java Applet feature uses Sun's AppletViewer to execute applet projects. The AppletViewer does not provide an applet execution environment similar to that provided by a production Web browser. As such, it is not suitable for production testing of applet projects.

In order to test your applet in a production-quality execution environment, you will have to create a Web application and then manually copy the applet classes into that project.

Application Project Conversion

When you import application projects into the SAS AppDev Studio 3.2, the source files are modified to remove the bookmark artifacts that were required by the source editor in webAF 3.0 software. In addition, two JAR files (`sas.ads.core.jar` and `sas.ads.misc.jar`) are added to the classpath that is defined by the SAS Versioned Jar Repository. These JAR files are added because they contain component classes that were used when webAF software generated the application code. At this time, SAS is deprecating the use of the classes contained in these two JAR files in favor of newer components. Since these JAR files might not be available in future versions of SAS AppDev Studio, you should discontinue using them.

When you created an application project in webAF 3.0 software, two Java classes -- `<projectName>.java` and `<projectName>Main.java` were created by default. The application implementation code resided in the `<projectName>Main.java` class, while the application entry point was implemented in the `<projectName>.java` class. In order to test your application within the plug-ins, you will need to run the `<projectName>.java` class.

Web Application Project Conversion

Caution: Before importing a webAF Web application project, make sure that you are using a workspace path that does not include spaces. You might encounter problems testing a Web application if the workspace path contains spaces.

To migrate a webAF Web application project, you simply import the project using the webAF Project Import feature by selecting **File > Import > webAF Project Import** from the menu bar. The import process automatically converts and upgrades the project to a SAS Web Application Project. This project type is based on the Dynamic Web Project provided in the Eclipse Web Tools Platform. Because the Eclipse Web Tools Platform defines specific locations where content under development must be stored, the conversion from a webAF 3.0 Web Application project to a SAS Web Application project in SAS AppDev Studio 3.2 involves changes to your Web application project structure.

First, the JAR files included in your Web application are handled as follows:

- Certain SAS common JAR files will not be copied from the webAF project. Instead, they will be replaced with references to the JAR files in the SAS Versioned Jar Repository by adding them as dependencies in the project's J2EE Module Dependencies list.[2]

- All other JAR files will be copied to the `/WEB-INF/lib` directory under the specified Web Content Folder for the Eclipse project.

The migration process will copy Java source files to the new project using the following rules:

- Java source files found in the webAF project's `WEB-INF/classes` directory will be copied to the specified Java Source Folder for the Eclipse project. The associated class files are not copied since they will be built by Eclipse and included in the Web application's `WEB-INF/classes` directory. During this process, any Foundation Services context listener classes that are detected will be replaced with an upgraded version that uses newer support classes provided in SAS 9.1.3 SP4. Additionally, any Build Frame servlets that are copied will be updated to remove comment blocks that were required by the webAF 3.0 source code generator. These changes are in code comments only and have no impact on the content of the compiled Java class files.

- Java source files found in the content portion of the Web application in the webAF project will be copied to the specified WebContent folder within the Eclipse project. Associated class files are not copied. Note that Java files stored in the WebContent folder will not be visible in Java-aware navigator views such as the Package Explorer or Project Explorer. They will be visible in the simple Navigator view. In addition, corresponding class files will not be built and included in the Web application content. This is because the Eclipse Web Tools Platform currently does not support building Java classes (such as applet classes or Java Web Start classes) in the content portion of a Web application[3].

- Java files found in the webAF project, but outside of the Web application, will be copied to the same relative path under the root of the Eclipse project. Associated class files are not copied. Like Java source files under the content portion of the Web application, these are not in a specified Java source folder, so they will not be visible in Java-aware navigator views and will not be built into

[2] The SAS Repository includes Log4J versions 1.2.3 and 1.2.9. The import process incorrectly uses version 1.2.9 of Log4J when it should use 1.2.3. You can use the SAS Repository Configuration dialog to change the version of Log4J to 1.2.3.

[3] A workaround is described in the "Building Java Classes into the Content Portion of a Web Application Project" document, which is available from the SAS AppDev Studio Developer's Site (support.sas.com/rnd/appdev/).

classes. If these Java files are built into a JAR file that is used by the containing Web application project, then it is best to move the classes into a separate Java project. Then, to use the output of this new Java project in your Web application, open the Properties dialog box for the SAS Web Application project and use the J2EE Module Dependencies page to add the separate project as a dependency.

Next, static Web application content (such as HTML files, images, and so on) will be migrated as follows:

- Static content in the webAF project that is part of the Web Infrastructure Kit will not be copied to the Eclipse project. Instead, the static content for the SAS 9.1.3 SP4 version of the Web Infrastructure Kit is copied into to the Eclipse project.
- All other Web application content is copied to the WebContent folder of the Eclipse project using the same relative path within the Web application.

The remaining files in a webAF Web application project are copied as follows:

- Files that are located under the project folder but outside of the Web application content area will be copied to the same relative location under the imported project. Should that location happen to be under the `build/classes` directory, then these files should be moved to a different folder. This folder is the default Java output folder whose contents will be deleted if you clean up a project by selecting **Project > Clean**.
- External files that have been inserted into the webAF project will be copied into the imported project under a folder named `external_files` with subfolders that duplicate the absolute path of the file from the root of the drive.

During the migration process, a file called `launchParameters.txt` will be created in the root folder for the project. This file contains the Java startup arguments needed by the project. If you defined custom server startup arguments in your webAF Web application project, some of those arguments will be included for you automatically, as follows:

- Java heap size arguments will be the greater of the SAS AppDev Studio 3.2 Web application template defaults or those specified in the imported project.
- Values for known server-related system properties will be imported.

You will have to manually transfer any additional custom arguments or changes that you made to the startup arguments in your webAF Web application project when you migrate it. If you later update this new project with content from a Web application template, the `launchParameters.txt` file might get overwritten. If you have custom startup arguments, you might want to save them in a separate file.

As a final step in the conversion process, additional metadata is created for the project to support its use of the SAS Repository and for compatibility with the SAS AppDev Studio 3.2 Web application templates.

Project Deployment Considerations

Before you deploy any applications created with the SAS AppDev Studio 3.2 Eclipse Plug-ins, there are a number of important concepts that should be understood. The following sections compare how things worked in previous versions of SAS AppDev Studio to how they work in the new release.

SAS Versioned Jar Repository

In past releases, the SAS AppDev Studio install copied SAS library JAR files to multiple locations, sometimes creating duplicate copies, in order to support different usage scenarios such as application development, Web application development, and JSASNetCopy downloads. The project classpath was configured in the Project Properties dialog box; either the SAS AppDev Studio ClassLoader was turned on or items were added to the classpath manually. In the case of Web application projects, the SAS library JAR files were also copied into the Web application file structure under each project. These limitations made it difficult to develop against different versions of the same JAR file or to know which JAR files were not needed for a given project.

One of the goals of the new SAS Versioned Jar Repository is to eliminate the duplication of JAR files. The new SAS Java Project type allows a project's classpath to reference SAS Versioned Jar Repository content to provide for the easy building and testing of applications which use common libraries from SAS. The SAS Repository Jar Selector provides the ability to specify the project classpath as a set of dependencies on common JAR files that are stored in the SAS Versioned Jar Repository. The project classpath can be configured as a particular subset of JAR files or to target development against specific versions of JAR files. You can now develop projects that reference a small set of the SAS library JAR files instead of the entire library. The Organize SAS Imports action and the SAS Repository Quick Fix feature automatically manage your project dependencies by adding import statements to your Java source files and by updating the project classpath as you use SAS library classes in your code.

When the time comes to deploy an application, you can use the application export dialog box to copy the specific SAS library JAR files that the project uses from the SAS Versioned Jar Repository to the export area. Or, if the SAS Versioned Jar Repository is already installed on the destination machine (see “Installing the SAS Versioned Jar Repository” below), the export wizard has options for allowing exported applications to reference the SAS Versioned Jar Repository directly.

SAS Launcher

The SAS Launcher is a custom ClassLoader that is installed as the Java application ClassLoader. Various SAS applications use the launcher (possibly with an .exe wrapper that displays a splash screen) to isolate the application classpath from JAR file conflicts (for example, it limits visibility into the Java Runtime Environment extensions directory). The SAS Launcher also provides a few convenience features like being able to specify the directories of JAR files to add to the application classpath.

In SAS AppDev Studio 3.2, the SAS Launcher has been updated to use the SAS Versioned Jar Repository, which enables applications to use a simple configuration file to add SAS Versioned Jar Repository content to their application classpath. The SAS Launcher is now also available for use by applications developed within SAS AppDev Studio.

Using the SAS Launcher typically requires adding several command-line parameters to the invocation of a Java program. Java system property definitions are used to get the custom ClassLoader installed as the Java application ClassLoader, and to optionally specify a configuration file requesting content from the SAS Versioned Jar Repository. In addition, the classpath defined on the Java command line is used for

loading the custom ClassLoader class. That classpath must therefore include a reference to `sas.app.launcher.jar` that is installed in the SAS Versioned Jar Repository and rather than mixing it with the real application classpath, the classpath for the launched application is defined in a separate system property. The SAS Java Project Export dialog box can generate a batch file containing a command line with all of the necessary parameters when exporting an application.

SAS Repository Locator

The SAS Repository Locator provides an alternative to the extended Java command line that is normally required to use the SAS Launcher. Using the SAS Repository Locator, is it possible to preserve the `java -jar <application jar>` semantics, including being able to double-click a JAR file in the Windows Explorer to launch an application.

The SAS Repository Locator is an optional part of the SAS Java application export process. If you select the SAS Repository Locator option for an export, then the SAS Repository Locator code is added to the application JAR file. The JAR file manifest is also updated so that the SAS Repository Locator code is invoked when the application JAR file is executed. Additionally, any JAR files in the SAS Versioned Jar Repository that the application depends on are recorded in the application JAR file manifest.

As a result, when the application JAR file is run with `java -jar` (for example, by double-clicking in the Windows Explorer), and the SAS Repository Locator main method is run. The SAS Repository Locator code determines the location of the SAS Versioned Jar Repository on the machine and of the sas.app.launcher.jar within it. The SAS Launcher custom ClassLoader from that JAR file is created and then used to replace the original Java application ClassLoader. The SAS Launcher code then loads and runs the original application, using any recorded dependencies on SAS Versioned Jar Repository content.

Thus, the SAS Repository Locator enables applications that would normally be able to use the simple `java -jar` invocation pattern to keep that simplicity, while also making use of both the SAS Versioned Jar Repository on the installation machine and the SAS Launcher.

Exporting Application Projects

In the past, SAS AppDev Studio supported exporting Java applications via the Package Wizard. This wizard enabled you to select which resources you wanted to include in the exported application and then it built the application JAR file. Then it was your responsibility to handle the remainder of the deployment process.

The SAS AppDev Studio 3.2 Eclipse Plug-ins leverage the new SAS Versioned Jar Repository to make the deployment process easier. Similar to the Package Wizard, you can use the Eclipse Export dialog box to export your SAS Java Project. In this dialog box, you select which project resources should be included in the application JAR file. You can also choose whether or not to deploy the application against the SAS Versioned Jar Repository. The default is to deploy against the SAS Versioned Jar Repository. If you use the default, then you will be able to take the application JAR file and copy it to any other machine that has the SAS Versioned Jar Repository installed. For more information, see “Installing the SAS Versioned Jar Repository” below. When executed, the application JAR file will automatically locate the SAS Versioned Jar Repository and give your program access to the SAS libraries. However, if you do not use the default, the required library JAR files will be copied to the same location as your application JAR file. Then you can copy all the resulting JAR files can to the deployment location for execution.

Installing the SAS Versioned Jar Repository

You can install and use the SAS Versioned Jar Repository on remote machines for use by the SAS Application Launcher and the SAS Repository Locator.

Note: You must have administrator or root privileges to install the SAS Versioned Jar Repository successfully.

Follow these steps to install the repository:

1. Copy the SAS Versioned Jar Repository from the development machine where SAS AppDev Studio is installed to the target machine. You can find the SAS Versioned Jar Repository underneath the main SAS AppDev Studio install directory. The default location is `C:\Program Files\SAS\SASAppDevStudio\3.2\VersionedJarRepository.`
2. Open a command prompt on the target machine.
3. Navigate to the `eclipse\plugins` directory under the `VersionedJarRepository` root directory.
4. Execute the `java –jar sas.app.launcher_3.2.0.jar -install` command to install the SAS Versioned Jar Repository.

Exporting Web Application Projects

In webAF software, the Package Wizard created WAR files for Web applications. In the SAS AppDev Studio Eclipse Plug-ins, this is handled by the Eclipse Web Tools Platform extension to the Eclipse Export feature.

Follow these steps to create the WAR file:

1. Select **File > Export** to open the Export dialog box.
2. Select **WAR file** and then click **Next**.
3. Select the Web Module you want to use and then specify the Destination.
4. If needed, select the **Export source files** and **Overwrite existing file** check boxes.
5. Click **Finish**.

This WAR file can be deployed to any application server or servlet container or both that provides full support for the Servlet 2.3/JSP 1.2 standard using JDK 1.4.0 or higher. However, only certain combinations of the system hardware, the JDK, and the server have been tested by SAS and are considered supported for Web applications from SAS Web Application Projects.

If you are deploying to a production environment, you should consult the *Third Party Software for SAS 9.1.3 Foundation with Service Pack 4* document (support.sas.com/documentation/configuration/thirdpartysupport/v913sp4/thirdparty913sp4.html), specifically the JDK and Application Servers sections, for details about the supported combinations and the server security fixes that must be installed. Additional Web application deployment information is available on the SAS AppDev Studio Developer's Site (support.sas.com/rnd/appdev).

Exporting Portlet Projects

In webAF software, the project build script contained a "package-par" rule that built the Portlet Archive (PAR) file. In the SAS AppDev Studio Eclipse Plug-ins, the export process is now handled by an Ant build file in the project. This build file is created for the user either by the new content template wizard when a

new portlet is added to a project or by the webAF import wizard when an existing webAF portlet project is imported.

Follow these steps to create the PAR file:

1. Open the Navigator view.
2. Select the `{portlet-name}ParBuild.xml` file at the top level of the project.
3. Select the file entry. Open the pop-up menu and select **Run As > Ant Build**.

This Ant build script will package any generated classes and static content files into an archive file in the root of project.

Note: You might have to refresh the Navigator view to get Eclipse to notice the new file.

If you need to add any additional content to the archive, simply edit the Ant XML document and specify the other content that you want to include.

To deploy the completed archive file, copy the output file to your Portal deployed portlets directory.

Running SAS AppDev Studio 3.0 Projects Against a SAS 9.1.3 Foundation Server with Service Pack 4

There are cases where you might want to continue to run projects that were created using SAS AppDev Studio 3.0 against a SAS 9.1.3 Foundation Server with Service Pack 4 without modifying them. For example, if you have a large number of working projects and do not want to convert all of them *en masse* at the same time that you migrate forward to SAS 9.1.3 Service Pack 4.

However, there are several important issues that must be considered in such a scenario:

1. The SAS 9.1.3 Service Pack 4 server must be updated with the SAS AppDev Studio 3.0 Server-Side Catalogs.

2. If the SAS AppDev Studio 3.0 project classes call any custom server-side SCL classes, then you will need to make sure to deploy the custom SCL classes to the SAS 9.1.3 Service Pack 4 server.

3. In order to exploit the new SAS 9.1.3 Service Pack 4 server capabilities, SAS AppDev Studio 3.0 projects must be upgraded to use the SAS AppDev Studio 3.2 component library.

Note: SAS does not support the practice of running SAS AppDev Studio 3.0 projects and reports against SAS 9.1.3 Service Pack 4 servers. If you encounter problems while attempting to do so, you must first migrate your SAS AppDev Studio 3.0 project (or report) to SAS AppDev Studio 3.2. If problems persist after converting to the latest software levels, then you should contact SAS Technical Support for assistance.

SAS Server Connections and Scalability Features

In SAS AppDev Studio 3.2, the SAS Connect Driver for Java protocol (commonly known as SAS JConnect) is no longer supported. Applications developed using SAS AppDev Studio 3.2 can access programs and data on SAS servers using either of the following methods:

- The Integrated Object Model (IOM) protocol provided by SAS Integration Technologies
- Java Database Connectivity (JDBC) using the IOM JDBC driver or the SAS/SHARE JDBC driver.

The SAS AppDev Studio Middleware Server (MWS) is no longer provided. The MWS now has Level C support from Technical Support. For more information about levels of support, see the Technical Support Web site (support.sas.com/techsup/support.html#non_current). Instead of MWS, you should use the scalability features (connection pooling and load balancing) provided by SAS Integration Technologies. For more information on configuring and using pooling and load balancing, see the *SAS 9.1.3 Integration Technologies Server Administrator's Guide* (support.sas.com/rnd/itech/doc9/admin_oma/sasserver/poollb/index.html).