



SAS[®] AppDev Studio[™] 3.0

Migration Guide

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2003. *SAS® AppDev Studio™ 3.0 Migration Guide*. Cary, NC: SAS Institute Inc.

SAS® AppDev Studio™ 3.0 Migration Guide

Copyright © 2003, 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.

1st printing, December 2003

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 trademarks of their respective companies.

Table of Contents

Table of Contents	2
Introduction	4
Changes to the SAS AppDev Studio System Requirements	4
Supported Versions of SAS Software	4
Supported Operating Systems.....	4
Java Platform Requirements.....	4
Running AppDev Studio 2.0 Projects and Reports with SAS 9.1	5
Communication Protocols	5
Scalability Features	5
SAS Integration Technologies Connection Pooling	6
SAS Integration Technologies Load Balancing.....	7
Elimination of SASNetCopy	7
JSASNetCopy Enhancements	8
SAS AppDev Studio “Unplugged”	8
Applet Deployment Using JSASNetCopy	9
Supporting Execution of AppDev Studio 2.0 Applet Content Using JSASNetCopy	9
Build Environment Changes	10
Project Conversion Considerations for webAF Users	11
General Project Conversion Considerations	11
Project Conversion Scenarios	11
Applet Project Conversion	12
Application Project Conversion	12
JSP and Servlet Project Conversion.....	13
Java Component Library Changes	15
Jar File Changes	15
JSP Custom Tag Library Changes	16
Component Palette Changes in webAF 3.0	17

Introduction

SAS AppDev Studio 3.0 contains many new features that benefit SAS application developers. Chief among them are the following:

- Support for the latest Java standards including Java 2 Standard Edition 1.4 (J2SE 1.4) and Java 2 Enterprise Edition 1.3 (J2EE 1.3)
- J2EE compliant Web application development support, including support for project templates, Java Standard Tag Library (JSTL), Jakarta Struts, and generation of deployment descriptors
- Support for SAS 9.1 servers
- Integration with the SAS Foundation Services
- SAS Information Delivery Portal integration, including the ability to develop and deploy portlets to the SAS Information Delivery Portal
- Integration with the open-source standard Jakarta Ant build tool
- Team-based development support, including integrated support for version control systems that conform to the Microsoft SCC interface
- Improved product accessibility.

The purpose of this document is to provide users of prior versions of SAS AppDev Studio 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, and a list of obsolete or significantly changed functionality.

Changes to the SAS AppDev Studio System Requirements

Supported Versions of SAS Software

SAS AppDev Studio 3.0 now supports SAS 9.1 software, as well as SAS Release 8.2. SAS Release 6.12 servers are no longer supported. As a result, developers creating applications that target SAS Release 6.12 servers should not upgrade to SAS AppDev Studio 3.0 but instead continue to use AppDev Studio 2.0.3.

Supported Operating Systems

The SAS AppDev Studio development bundle can now be installed on the Windows XP Professional operating system. Windows 2000 Professional and Windows NT Workstation continue to be supported platforms, as well. Windows 95 and Windows 98 are no longer supported.

Java Platform Requirements

The SAS AppDev Studio development bundle requires Java 2 Standard Edition (J2SE) 1.4.1 or higher for both application development and execution of applications or applets at run time. For execution of Web applications, a servlet container supporting the Java Servlets 2.3 and JavaServer Pages 1.2 specifications is also required.

See the *AppDev Studio Java Components 3.0 System Requirements* document for complete system requirements information. A hard copy of the document is provided in the product packaging and it can also be viewed online at the SAS AppDev Studio Developer's Site (<http://support.sas.com/rnd/appdev>).

Running AppDev Studio 2.0 Projects and Reports with SAS 9.1

There are cases where you might want to continue to run projects and reports that were created using SAS AppDev Studio 2.0¹ against a SAS 9.1 server without modifications (for example, when you have a large number of working projects and reports and do not want to convert all of them *en masse* at the same time that you migrate forward to SAS 9.1). However, there are several important issues that must be considered in such a scenario:

1. The SAS 9.1 server must be updated with the SAS AppDev Studio server-side catalogs in order to support the project or report.
2. If the AppDev Studio 2.0 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 server.
3. In order to exploit new SAS 9.1 server capabilities (for example, the new scalable OLAP Server), AppDev Studio 2.0 projects must be upgraded to use the SAS AppDev Studio 3.0 component library.

Note that the practice of running AppDev Studio 2.0 projects and reports against SAS 9.1 servers is not supported. If you encounter problems attempting to do so, you must first convert your AppDev Studio 2.0 project or report to SAS AppDev Studio 3.0. If problems persist after converting to the latest release of the software, then you should contact SAS Technical Support for assistance.

Communication Protocols

In prior versions of AppDev Studio, the default communication protocol used by the Connection object (`com.sas.rmi.Connection`) was SAS Connect Driver for Java protocol (commonly known as JConnect). However, starting with SAS AppDev Studio 3.0, the default communication protocol used by the Connection object is the Integrated Object Model (IOM) protocol provided by SAS Integration Technologies.

You can continue to use the JConnect protocol for existing applications or new development, but be advised that at some point in the future support for JConnect may be eliminated from SAS AppDev Studio.

Scalability Features

Scalability features such as load balancing of SAS servers and maintaining pools of pre-initialized SAS servers to support remote client execution are provided by both the SAS AppDev Studio Middleware Server (MWS) and SAS Integration Technologies.

Although the MWS supports clients using the IOM protocol provided with SAS Integration Technologies, its scalability features predate SAS Integration Technologies and were mainly designed to support remote clients using the JConnect protocol. Since SAS Integration Technologies includes built-in support for scalability, it is recommended that you use these built-in features when communicating over a connection that uses the IOM protocol.

¹ Note that wherever this document refers to AppDev Studio 2.0, it also refers to any of the AppDev Studio 2.0 updates, as well, unless specifically stated otherwise.

You can still continue to use the MWS for existing applications or new development, but be advised that the MWS is being deprecated in favor of the scalability features provided by SAS Integration Technologies. Therefore, if you are using the IOM protocol to support communication from your remote clients, we highly recommend that you use the SAS Integration Technologies load balancing and workspace pooling support instead of using the similar features provided by the MWS.

The MWS runs as a standalone mid-tier application and provides basic load-balancing capabilities. Setting `rolloverType` in the `RocfORB.properties` configuration file enables the MWS to connect clients to servers in either a depth-first or breadth-first fashion. Client connections are allocated across one or more servers (each with a defined capacity) as listed in the `funnel.cfg` configuration file. The MWS also provides a basic form of connection pooling as specified by the `preloadedClients`, `autoRestart`, and `retain` properties in the `RocfORB.properties` file.

SAS Integration Technologies load balancing and connection pooling support is not implemented as a single, mid-tier application. Connection pooling and load balancing are separate and mutually exclusive. SAS Integration Technologies connection pooling is setup and run as part of the Java client. SAS Integration Technologies load balancing is setup through a SAS metadata server and runs on the backend servers. (NOTE: SAS Integration Technologies load balancing is available in SAS 9 and later). The SAS Integration Technologies *Choosing Pooling or Load Balancing* document² describes the basic considerations that need to be taken into account when choosing between connection-pooling and load balancing strategies.

SAS Integration Technologies Connection Pooling

SAS Integration Technologies connection-pooling parameters can be set on the IOM tab of the Edit Connection window or the New Connection window. Select **Pooling** from the Type of Workspace Factory pull-down menu. The following pooling settings are available:

Max Connections to Server

The maximum number of connections this factory should have to this server at any given time. If no value is specified, then the number of connections will be limited only by the physical capacity of the server.

Min Connections in Factory

The minimum number of live connections that must be associated with this factory at any given time unless the factory has been shut down or destroyed (includes both connections that have been allocated to users and unallocated connections). The default value is 0.

Min Unallocated Connections

The minimum number of unallocated live connections that must be associated with this factory at any given time unless the factory has been shut down or destroyed. The default value is 0.

Enable Connection Timeouts

When this option is selected, unallocated live connections will be disconnected after a period of time (determined by the value of the *Timeout (in minutes)* setting), unless they are allocated to a user before that period of time passes. Otherwise, unallocated live connections will remain alive indefinitely.

² http://support.sas.com/rnd/itech/doc9/admin_oma/sasserver/poollb/choosepoolb.html

Timeout (in minutes)

The timeout setting specifies the period of time, in minutes, that an unallocated live connection will wait to be allocated to a user before shutting down. This property is ignored if *Enable Connection Timeouts* is not selected. The value must not be less than 0, and it must not be greater than 1440 (the number of minutes in a day). The default value is 3. If the value is 0, then a connection returned to a pool by a user will be disconnected immediately, unless another user is waiting for a connection from the pool.

Connection pooling parameters may also be set in a `java.lang.properties` object and passed to the `com.sas.rmi.connection` object in client code as in this code sample:

```
java.lang.properties poolProperties = new java.lang.properties();
poolProperties.put("sasMaxPerWorkspacePool", 10);
poolProperties.put("sasMinSize", 5);
poolProperties.put("sasMinAvail", 3);
poolProperties.put("sasServerConnectionTimeout", "true");
poolProperties.put("sasServerConnectionTimeout", 5);

com.sas.rmi.connection connection = new com.sas.rmi.connection();
com.sas.rmi.Connection.setServerArchitecture(connection, "IOM");
connection.setIomWorkspaceFactoryType("pooling");
connection.setIomWorkspaceFactoryPoolingInfo(poolProperties);
```

This example is equivalent to using the following pooling settings in the connection customizer:

Max Connections to Server	10
Min Connections in Factory	5
Min Unallocated Connections	3
Enable Connection Timeouts	checked
Timeout (in minutes)	5

SAS Integration Technologies Load Balancing

SAS Integration Technologies load balancing can distribute client connections to multiple SAS sessions which can be on the same or different hosts. SAS Integration Technologies load balancing uses a cost algorithm to determine which SAS session will serve an incoming client connection. A description of the cost algorithm and load-balancing setup can be found in the SAS Integration Technologies *Planning for Load Balancing Metadata* document³.

Elimination of SASNetCopy

In AppDev Studio 2.0, two different technologies were provided to support automatic download and installation of JAR files to client machines running applets – SASNetCopy and JSASNetCopy. SASNetCopy is an older technology that uses either an ActiveX control under Microsoft Internet Explorer or Netscape's SmartUpdate technology. JSASNetCopy is a newer, pure Java technology that is browser neutral.

³ http://support.sas.com/rnd/itech/doc9/admin_oma/sasserver/iombridge/loadset.html

With the release of SAS AppDev Studio 3.0, SASNetCopy is no longer provided or supported. All of the SASNetCopy capabilities are provided by JSASNetCopy, which is a more complete, robust, and reliable automatic software installation solution. Both webAF and webEIS will automatically convert any existing projects that use SASNetCopy to use JSASNetCopy when they are migrated to SAS AppDev Studio 3.0.

If you customized your SASNetCopy solution to include additional JAR files to augment the default set of SAS JAR files, then you will need to make similar customizations to JSASNetCopy. For information on how to customize JSASNetCopy, see the *JSASNetCopy Detailed Description*⁴ and *JSASNetCopy Administration*⁵ documents on the SAS AppDev Studio Developer's Site.

JSASNetCopy Enhancements

SAS AppDev Studio “Unplugged”

In AppDev Studio 2.0, the SASNetCopy and JSASNetCopy automatic installation technologies both relied upon use of the Java Runtime Environment's (JRE) extensions directory to support using downloaded JAR files without requiring classpath or Java policy file modifications on the client machines. By downloading JAR files to the extensions directory, they were made available to all applets and applications using the associated JRE. This was advantageous in that it provided a zero-install solution for Java applet execution. However, this approach had several associated disadvantages as well:

1. Due to the shared nature of the extensions directory, all Java applications or applets using the JRE could potentially load classes from the downloaded JAR files. Since the extensions classloader has precedence over applet classloaders, there was no way for deployed applets to override JAR files located in the extensions directory (for example, if the applet needed to run on an earlier version of a JAR file supplied by SAS or by a third party).
2. Applets or applications running concurrently all had to use the same versions of the downloaded JAR files.
3. Due to default privileges assigned to any classes loaded from the extensions directory, unsigned content downloaded into the extensions directory ran with privileges usually only accorded to signed content.
4. Deleting obsolete JAR files from the extensions directory was problematic, because if a Java application were running at the same time, then all JAR files in the extensions directory were locked.
5. When updating the Java Development Kit (JDK) on their local machine, developers using AppDev Studio had to copy the AppDev Studio JAR files under the new JDK installation in order for AppDev Studio to function properly with the new JDK.

In order to overcome these restrictions, the version of JSASNetCopy delivered with SAS AppDev Studio 3.0 has been designed to maintain a version-based JAR repository separate from the extensions directory. When applets are executed using JSASNetCopy, any required JAR files are downloaded to this local JAR repository instead of to the JRE's extensions directory. Then, JSASNetCopy creates a special classloader

⁴ <http://support.sas.com/rnd/appdev/Tools/jsasnetcopy-details.htm>

⁵ <http://support.sas.com/rnd/appdev/Tools/jsasnetcopy-administration.htm>

for the applet that includes specific versions of required JAR files, loaded in-memory from the JAR repository. Since each applet gets its own in-memory classloader, there is no conflict if concurrently running applets require different versions of a given JAR or a set of JAR files. Additionally, since the content is not downloaded to the extensions directory, JSASNetCopy can properly manage the security context for unsigned content, ensuring that it does not run with enhanced privileges.

Applet Deployment Using JSASNetCopy

In AppDev Studio 2.0, the JSASNetCopy Web application was deployed as part of the developer site update install program (Update a Web Server). This process copied all of the JSASNetCopy files and downloadable SAS JAR files, as well as developer information (such as Help documents, sample projects, and so on) to the root directory of a pre-existing Web server. This worked well enough for a developer machine, but posed various problems for administering a production environment.

Now, with SAS AppDev Studio 3.0, all of the content necessary to make use of JSASNetCopy -- the JSASNetCopy code, configuration files, and the JAR files available for download -- is supplied as a single Web application in a Web application archive (.war) file format. For easy migration, the JSASNetCopy Web application includes AppDev Studio 2.0 libraries in order to support existing AppDev Studio 2.0 applets running in Java 1.3. The SAS AppDev Studio 3.0 libraries will support both AppDev Studio 2.0 and SAS AppDev Studio 3.0 applets when running in Java 1.4 or later.

Customizing which JAR files JSASNetCopy will download is also easier. By adding or updating the JAR files in the <ADS install>\Java\JSASNetCopy\webapp\jars directory and running an update tool in webAF (by selecting **Tools > Update JSASNetCopy .war** from the menu bar), a developer can create an updated JSASNetCopy Web application archive located at <ADS install>\Java\JSASNetCopy\JSASNetCopy.war.

Even for Web servers without a Web application container, the JSASNetCopy Web application archive can be useful as a single archive file which contains all of the necessary content. Currently, none of the JSASNetCopy Web application content is dynamic, so extracting the JSASNetCopy.war file to suitable location on the Web server is an alternate method of deploying JSASNetCopy resources.

Segregating the JSASNetCopy resources from the developer materials saves space on the Web server and has the added benefit of eliminating the need for an external Web server on the AppDev Studio development machine. The SAS AppDev Studio Developer's Site and other materials are now made available using the version of Tomcat included in the SAS AppDev Studio install.

For more information on JSASNetCopy deployment, see the *Deploying with JSASNetCopy* and *JSASNetCopy Administration* documents on the SAS AppDev Studio Developer's Site.

Supporting Execution of AppDev Studio 2.0 Applet Content Using JSASNetCopy

As an organization moves from AppDev Studio 2.0 to SAS AppDev Studio 3.0, it is recommended that Web servers that supply JSASNetCopy resources be upgraded to serve the SAS AppDev Studio 3.0 JSASNetCopy content, as doing so should enable applets developed with AppDev Studio 2.0 to run in the widest variety of situations.

The ability of HTML to specify which version of the Java Plug-in (implemented as an ActiveX control for Internet Explorer or a plug-in for Netscape) to invoke is, unfortunately, very limited and somewhat inconsistent across the two platforms. Suffice it to say that some HTML pages might invoke a specific version of the Plug-in (such as Java 1.3.0_01), whereas other pages might be written to use any version of

Java that is 1.3 or better. By default, SAS AppDev Studio 3.0 will generate HTML which requests Java 1.4 or later. However, older applets developed with versions of AppDev Studio prior to 3.0 might be using a variety of specifications. The end result is that client machines might have more than one version of Java and different clients running the exact same applet might use different versions of Java to do so.

This presents a difficulty when trying to support applets developed with AppDev Studio 2.0 that have not been converted to SAS AppDev Studio 3.0. While the SAS AppDev Studio 3.0 libraries are backward-compatible in terms of API usage, they are targeted at Java 1.4 and might require features only found in Java 1.4 or later. On the other hand, Java 1.4 does not support the existing AppDev Studio 2.0 libraries due to incompatible changes in the Java environment itself.

JSASNetCopy provides at least a partial solution to this migration dilemma. When invoked, JSASNetCopy detects whether it is running in Java 1.3 or Java 1.4. By including the AppDev Studio 2.0 libraries with the JSASNetCopy 3.0 resources, JSASNetCopy is able to download the AppDev Studio 2.0 libraries when running in a version of Java prior to 1.4 and download the SAS AppDev Studio 3.0 libraries when running in Java 1.4 or later. Because the SAS libraries are API-compatible, as long as an AppDev Studio 2.0 applet does not have conflicts of its own with Java 1.4, and there are no other environmental barriers, it should be able to run using the SAS AppDev Studio 3.0 libraries. Of course, applets developed with SAS AppDev Studio 3.0 will require a Java 1.4 environment and HTML that specifies the need for Java 1.4 or later.

Build Environment Changes

In SAS AppDev Studio 3.0, webAF uses the open source Apache Ant⁶ build tool to compile user projects, create the WAR file for Web application projects, and perform various internal tasks. Using Ant to manage builds provides a dramatic compilation performance improvement over webAF 2.0, especially for projects containing many files. SAS AppDev Studio ships with an embedded Ant installation to provide this support.

When you first build a webAF project, webAF generates a `<project-name>_build.xml` file that describes what Ant needs to do to build the project. Each time you build your project, webAF updates this file to contain the current classpath entries and the set of files to be built. The build file contains build “targets” that are various operations that support project builds. If you are familiar with Ant, you can modify the build file by changing any build target that is not marked as “auto generated” or by adding new targets. Targets whose name contains a leading underscore are internal, private targets used by webAF and should not be modified. Targets that are marked as “auto generated” are updated prior to each build based upon changes to the project contents or properties, application options, and so on. Such targets should not be edited directly. Since the Java compiler options are located in auto generated blocks, you must change the build options by selecting **File > Project Properties** or **Tools > Options** from the menu bar and then modifying the compiler options.

If you have an existing project that you build using Ant, you can import the project into webAF 3.0 by selecting **File > New** and then selecting **Create Project from Directory** on the Projects tab. Then, the project wizard will prompt you for the location of your project’s `build.xml` file and import your existing project based upon the `build.xml` file contents.

⁶ <http://jakarta.apache.org/ant>

You can execute any of the targets in the `build.xml` file by selecting **Build > Project Tasks** from the webAF menu bar. The Project Tasks menu contains a submenu listing all non-private targets in the `build.xml` file, including any custom targets that you have added.

Project Conversion Considerations for webAF Users

General Project Conversion Considerations

In order to open a webAF 2.0 project in webAF 3.0, you must convert the project files to the new version 3.0 file formats. You will be prompted as to whether or not you want to perform the conversion. If you choose to allow the conversion to proceed, a back-up copy of the original project files will be created. After the backups are created, the project file and any frame files will be updated to the new file formats. In addition, the Ant build file will be generated to support builds of the new project content. Once this conversion is completed, the updated files will be saved to disk and then code generation will be run to update the generated code to match the webAF 3.0 standards.

If you have not previously installed the AppDev Studio Java Components 2.0.3 Update, it is strongly recommended that you do so prior to installing SAS AppDev Studio 3.0. After applying the AppDev Studio Java Components 2.0.3 Update, any webAF projects that you plan to convert forward to webAF 3.0 file formats should be opened, rebuilt, and saved within webAF 2.0.3. Note that this is an absolute requirement for servlet projects due to changes in servlet code generation in webAF 2.0.3. After this, you can upgrade to SAS AppDev Studio 3.0.

The project conversion process can take some time, especially for larger projects or on slower machines with less available memory. If you encounter an out-of-memory condition while performing the conversion, then you will need to restore the saved backups of the original project and restart webAF with a larger Java heap size. For example, executing `webAF.exe -Xmx512m` will allocate 512 MB of heap space for the Java VM running within webAF.

If your webAF 2.0 project uses a JAR file that is located in the extensions directory, then you should temporarily place that JAR file in the `\java\sas\ext` subdirectory under the SAS AppDev Studio 3.0 root installation directory prior to conversion of the project. Otherwise, webAF 3.0 will not be able to see the JAR file during project conversion due to the new “unplugged” implementation. After the project conversion is completed, the JAR file should be moved under the project directory and added to the classpath of the project by selecting **File > Project Properties** from the webAF menu bar.

See below for details on specific project conversion scenarios.

Project Conversion Scenarios

In webAF 2.0, the major project types that were supported included applet, application, JavaServer Pages (JSP), and Java servlet projects. In webAF 3.0, the JSP and servlet project types have been combined into a Web application project type. This Web application project type mirrors a J2EE-style Web application that can be deployed to a standard J2EE application server.

The subsections below describe particular considerations for the various project conversion operations based upon the type of project being migrated from AppDev Studio 2.0.

Applet Project Conversion

Because SASNetCopy is no longer supported, webAF 2.0 applet projects that use SASNetCopy will be converted to use JSASNetCopy when migrated to webAF 3.0.

With the "unplugging" of SAS AppDev Studio from the extensions directory, the SAS AppDev Studio 3.0 install does not copy JAR files into the current versions of Java's extensions directory. In AppDev Studio 2.0, webAF executed applet projects at design-time using Java Plug-in HTML (without SASNetCopy or JSASNetCopy) with a `file://` URL. This was possible because all of the SAS libraries that an applet might need were already installed in the extensions directory on the development machine. However, this also meant that development testing was somewhat removed from real world deployment since using a `file://` URL causes the applet to run with different security settings than actually downloading and running an applet using an `http://` URL.

With the "unplugged" nature of SAS AppDev Studio 3.0, the old approach to executing applets no longer works. Consequently, webAF 3.0 uses JSASNetCopy to preview applets at design time. A private Tomcat server is started for applet projects similar to what is done for Web application projects. The applet is previewed using an `http://` URL which points to `localhost` and to the port for the private Web server. This means that the design-time preview is a more realistic test of deployed applet execution. It also means that the first time an applet is previewed with webAF 3.0; JSASNetCopy will need to auto-download any necessary SAS AppDev Studio 3.0 Java class libraries, just like it would on any non-developer client.

This more realistic approach to previewing applet execution has the following implication for unsigned applets. As in a production deployment environment, both the Web server and the SAS server must be running on the same machine or the Web server machine must run some sort of middleware, because unsigned applets can only make network connections to the host from which they are downloaded. During initial development testing, the developer's machine typically hosts both the Web server and the SAS server. For this reason, in SAS AppDev Studio 3.0, new Connection components have the "SAS server and Web server are same machine" property set to true by default. For webAF 2.0 applet projects or webEIS 2.0 reports being migrated forward, this property is set automatically during conversion, if necessary.

Application Project Conversion

Application projects that you convert to webAF 3.0 will have their starting parameters modified to use a new application launcher technology. This new technology was put in place to aid in project deployment and to keep webAF 3.0 developed applications "unplugged" from the extensions directory. The application launcher ignores most of the JAR files that are in the extensions directory, except for those that are part of the standard Java runtime distribution. This enables the application to use the JAR files installed as a part of SAS AppDev Studio 3.0 and not have them overridden by older versions possibly located in the extensions directory.

The `sas.launcher.jar` file contains the classes needed by the new launching technology. Behavior of the application launcher is controlled by the following three properties:

<code>sas.app.class.dirs</code>	Specifies a list of directories much like <code>java.ext.dirs</code>
<code>sas.app.class.path</code>	Specifies a list of JAR files and class directories much like <code>java.class.path</code>
<code>sas.java.config</code>	Points to a configuration file that lists the JAR files that should be read from the Java extensions directory. The default set of JAR files is <code>dnsns.jar</code> , <code>ldapsec.jar</code> , <code>localedata.jar</code> , and <code>sunjce_provider.jar</code> .

Assuming that SAS AppDev Studio is installed to the default location of `c:\appdevstudio`, the syntax for using the application launcher is:

```
java -classpath c:\appdevstudio\webaf\bootlib\sas.launcher.jar
-Dsas.app.class.dirs=c:\appdevstudio\java\sas\ext
-Dsas.app.class.path=<project-location>
-Djava.system.class.loader=com.sas.app.AppClassLoader
<application-main-class>
```

JSP and Servlet Project Conversion

In webAF 3.0, the JSP and Servlet project types have been replaced by the new Web Application project type. Opening a JSP or Servlet project that was created using webAF 2.0 will cause webAF 3.0 to display a dialog box asking for permission to convert the project to a Web Application project. You must accept the conversion to be able to open the project in webAF 3.0.

The behavior of the conversion dialog will differ slightly depending on whether you specified a Web Application Base Directory by selecting **File > Project Properties** and updating the Options tab (for example, the project was associated with a Web application). If you did not specify the Web Application Base Directory, then the dialog box presents a choice of **OK** or **Cancel**. Click **OK** to convert the project and cause a Web application to be created. It will be found under the `webapp` subdirectory of the project. Current project content will be backed up under the project in a subdirectory called `conversion_backup` and then moved into the newly created Web application. The files handled in this fashion include:

- Files with the extensions `jsp`, `html`, `htm`, `css`, `gif`, and `jpg` will be moved under the `webapp` directory.
- Files with the extensions `java` and `fml` will be moved under the `webapp\WEB-INF\classes` directory.

If you specified the Web Application Base Directory, then the conversion dialog box offers you a choice of moving the Web Application Base Directory under the project directory. If you select **Yes**, the Web application will be copied under the project to a subdirectory called `webapp`. Note that a back-up copy of the Web application will not be made since the original remains intact. If you select **No**, the entire Web application will be backed up to the `conversion_backup` subdirectory of the project and the project will continue to use the Web application at its current location. In either case, for Servlet projects, all files under the project with `java` or `fml` extensions will be backed up and then moved to the Web application under its `WEB-INF\classes` directory.

The conversion process will always create or update the Web application's deployment descriptor (`web.xml`) to webAF 3.0 standards. This includes changing the DOCTYPE to match the Servlet 2.3/JSP 1.2 requirements and performing other modifications that SAS AppDev Studio 3.0 requires. In addition, if the conversion detects that the SAS tag library was being used, additional deployment descriptor modifications are performed – including automatically adding servlet mappings for various servlets used by SAS AppDev Studio 3.0 components – and the Web application will be upgraded with the required JAR files and other content to support its continued use. If SAS tag library use is not detected, no additions are made to the Web application. Projects without a Web Application Base Directory specified are assumed to use the SAS tag library.

If you have an AppDev Studio 2.0 servlet or JSP project where the project's home directory (the directory that contains the project's `.afx` file) and the Web application base directory (the directory that contains the `WEB-INF` subdirectory) are the same directory, then it is recommended that you do not try to open this project such that webAF 3.0 automatically converts it to a Web Application project. Instead, you should manually convert the project using the following steps:

1. Create a new Web application project.
2. On the first wizard panel, select the **Use existing web application at specified location** radio button and then update the Web Application Location field with the path for the Web application that you want to convert.
3. Select the **Copy existing web application to project directory** check box. This enables webAF's project management files to be separated from the Web application files, while also keeping an old copy of your project intact.
4. Accept the default settings and continue to click the **Next** button, until you reach the Select Web Application Initial Content panel in the wizard. Hold the CTRL key and click to deselect the default JavaServer Page selection. This prevents any new content from being created.
5. Accept the remaining defaults in the wizard to finish creating the Web application project.
6. Delete the old `.afx` and `.wsp` files, as well as any other project-related files, such as backup files, from the Web application, which is now located in the `webapp` directory of the new project.
7. Inspect the `webapp\WEB-INF\lib` directory and remove any duplicate JAR files. For example, the old Web application might have contained `log4j.jar` and the update might have added `log4j-1.2.3.jar`. Only one of these files should be kept—probably the newest version.
8. Re-insert the files that you want into the new project. From the webAF menu bar, you can select **Insert > File Into Project** or you can select **Insert > Directory Into Project**.

At this point, you have an upgraded project that contains your prior Web application content.

Note that unlike webAF 2.0, webAF 3.0 implements the general Java best practice recommendation that Web applications be self-contained. As a result, all of the SAS JAR files that might be needed are copied to the Web application's `WEB-INF\lib` directory. For a Web application that uses the SAS tag library, this involves about 50 JAR files consuming about 30 MB of disk space. This is necessary to avoid potential deployment and versioning issues that arise when deploying these JAR files to a servlet container outside of the Web application.

Also note that prior to displaying the project conversion dialog box, the Web application's deployment descriptor is checked for errors. If any error is found, then the dialog box appears showing the error(s). All errors must be corrected before the project can be converted. Selecting

the **OK** or **Cancel** button cancels the conversion. For help correcting the errors, see the *Troubleshooting web.xml Errors*⁷ document on the SAS AppDev Studio Developer's Site.

Java Component Library Changes

JAR File Changes

In AppDev Studio 2.0 and earlier, most of the component classes that were available were shipped in the `webAF.jar` file. As a result, the size of this JAR file grew to around 10 MB. This caused the download time for applets to become prohibitive for some customers with slower speed connections (such as with dial-up access). To help alleviate this problem, the `webAF.jar` file was broken up into a number of smaller JAR files in SAS AppDev Studio 3.0. The JAR file split was based on dependencies between the classes and the functionality provided by the classes. In addition, the names of all of the JAR files that are shipped with SAS AppDev Studio 3.0 have changed to conform to new standards that avoid any naming conflicts with JAR files that our customers might already have in place.

In SAS AppDev Studio 3.0, the contents of what was formerly the `webAF.jar` file are now delivered in the following set of JAR files:

JAR Name	Purpose
<code>sas.ads.core.jar</code>	Low-level classes used by SAS AppDev Studio
<code>sas.ads.dt.jar</code>	Design-time classes such as property editors and customizers
<code>sas.ads.misc.jar</code>	Remote SAS access classes (for example, ROCF) and many other older components
<code>sas.core.jar</code>	Low-level classes used by many applications and components
<code>sas.icons.jar</code>	Icons used by visual components
<code>sas.storage.jar</code>	JDBC adapters and V9 OLAP model
<code>sas.swing.jar</code>	Swing-based components and utility classes.
<code>sas.swing.remote.jar</code>	Swing-based components which need to access remote models and services
<code>sas.text.jar</code>	Formatting classes that support SAS formats
<code>sas.webEIS.jar</code>	webEIS run-time classes

⁷ <http://support.sas.com/rnd/appdev/webAF/server/WebXmlErrors.htm>

The following JAR files changed names:

Old JAR Name	New JAR Name
webAFServerPages.jar	sas.ads.servlet.jar
queryAF.jar	sas.JQuery.jar
webEIS.jar	sas.webEIS.jar

JSP Custom Tag Library Changes

In AppDev Studio 2.0, all of the TransformationBeans along with the SAS Custom Tag library were contained in the `webAFServerPages.jar` file. In SAS AppDev Studio 3.0, the `webAFServerPages.jar` file has been replaced with two JAR files:

- `sas.servlet.jar`
This JAR file contains all of the new TransformationBeans that are located in the `com.sas.servlet.tbeans` package along with the corresponding custom tags located in the `com.sas.taglib.servlet.tbeans` package. In addition, this JAR file contains the common/base classes that are used by other classes in this JAR file along with `sas.ads.servlet.jar`. This JAR file will automatically be included in your `WEB-INF/lib` directory.
- `sas.ads.servlet.jar`
This JAR file contains the AppDev Studio 2.0 TransformationBeans that are located in the `com.sas.servlet.beans` package along with the corresponding custom tags located in the `com.sas.taglib.servlet.beans` package. This JAR file is dependent on `sas.servlet.jar`; so you will also need to include this JAR file in your `WEB-INF/lib` directory.

Another difference between the two releases pertains to where the Tag Library Definition (.tld) files are located. In AppDev Studio 2.0, the `sasads.tld` file for the SAS Custom Tag Library was loosely deployed in the `META-INF` directory of your Web application. The Web application's `web.xml` file also contained an explicit mapping for this TLD file that was used in the `taglib` directive. In SAS AppDev Studio 3.0, the TLD file is no longer loosely deployed but rather bundled within the JAR file. By bundling the TLD in the JAR file, there is no longer the need to map the TLD in the `web.xml` file. The servlet container will automatically discover and map the bundled TLD files.

The following are the two main SAS Custom Tag Libraries in SAS AppDev Studio 3.0. The prefixes used in the directives below are just the default prefixes that we suggest; they can be changed.

- `sasads.tld`
This tag library contains the same tag as it did with AppDev Studio 2.0 and is bundled in the `sas.ads.servlet.jar` file. The directive to access these tags is also the same as it was in SAS AppDev Studio 2.0:

```
<%@ taglib uri="http://www.sas.com/taglib/sasads" prefix="sasads" %>
```

- `sas.tld`

This tag library is new to SAS AppDev Studio 3.0 and is bundled in the `sas.servlet.jar` file. The tag library contains new tags in addition to tags that replace the tags in the `sasads.tld` file. The directive to access this tag library is:

```
<%@ taglib uri="http://www.sas.com/taglib/sas" prefix="sas" %>
```

The `sas.tld` tag library is also available in the following smaller split-up tag libraries. When you know that your particular Web application will only be dealing with a select few tags, you might want to use one of the smaller tag libraries.

- `sas-dv.tld`

This tag library contains data viewer tags such as the `TableViewComposite`. The directive to access this tag library is as follows:

```
<%@ taglib uri="http://www.sas.com/taglib/sas-dataviewers" prefix="sas-dv" %>
```

- `sas-f.tld`

This tag library contains form-related tags such as the `Form`, `ComboBoxView` and `ListBoxView` tags. The directive to access this tag library is as follows:

```
<%@ taglib uri="http://www.sas.com/taglib/sas-form" prefix="sas-f" %>
```

- `sas-g.tld`

This tag library contains SAS/GRAPH tags such as the `BarChart` and `PieChart` tags. The directive to access this tag library is as follows:

```
<%@ taglib uri="http://www.sas.com/taglib/sas-graphics" prefix="sas-g" %>
```

- `sas-ip.tld`

This tag library contains `iPage` tags such as the `IText` and `IMenu` tags. The directive to access this tag library is as follows:

```
<%@ taglib uri="http://www.sas.com/taglib/sas-ipage" prefix="sas-ip" %>
```

- `sas-s.tld`

This tag library contains selector related tags such as the `MenuBar` and `TreeView` tags. The directive to access this tag library is as follows:

```
<%@ taglib uri="http://www.sas.com/taglib/sas-selectors" prefix="sas-s" %>
```

- `sas-u.tld`

This tag library contains utility tags such as the `Label` tag. The directive to access this tag library is as follows:

```
<%@ taglib uri="http://www.sas.com/taglib/sas-util" prefix="sas-u" %>
```

Component Palette Changes in webAF 3.0

The component palette layout within webAF varies by project type. For a given project type, there may be several palettes available for use. The currently displayed palette is selectable via a drop-down list that is displayed in the title bar of the component palette window. In webAF 3.0, the following component palettes are available for applet and application projects:

AWT ⁸	Contains AWT-based visual components (set as the default palette for applet or application projects that use AWT).
Swing	Contains Swing-based visual components (set as the default palette for applet or application projects that use Swing).
All Models	Contains the various model components that connect to back-end data services.
User Palette	Contains user-defined components, code snippets, and so on.

In webAF 3.0, the set of palettes that is available when editing a Web application project depend upon the various tag libraries that are specified in the **New Web Application Wizard** or by selecting **File > Web Application Properties** from the webAF menu bar. The following palettes may be available, depending upon the specified Web application project options that you specify:

SAS JSP/Servlet (Version 3)	Contains the JSP custom tags and the TransformationBeans supplied by SAS for accessing and displaying data from server-side Java.
JSP Standard Tag Library – EL	Contains “expression language” JSP custom tags from the JSP Standard Tag Library. ⁹
JSP Standard Tag Library - RT	Contains “request-time expression language” JSP custom tags from the JSP Standard Tag Library.
SAS JSP/Servlet (Version 2) ¹⁰	Contains the JSP custom tags and the TransformationBeans supplied by SAS in AppDev Studio 2.0.
SAS MDDB JSP/Servlet (Version 2) ¹⁰	Contains the MDDB-related JSP custom tags and the TransformationBeans supplied by SAS in AppDev Studio 2.0.
All Models (Version 2) ¹⁰	Contains the InformationBeans and the utility classes supplied by SAS in AppDev Studio 2.0 that support access to the back-end SAS server.
User Palette	Contains user components, code snippets, and so on.

⁸ Note that AWT (Abstract Windows Toolkit) is older, deprecated technology. While you still might use AWT-based visual components within webAF, it is strongly recommended that you use the Swing-based visual components when building any new applet or application projects. The AWT-based visual components will likely be removed from SAS AppDev Studio in a future release.

⁹ See <http://java.sun.com/jstl> for more information on the JSP Standard Tag Library.

¹⁰ The *SAS JSP/Servlet (Version 2)*, *SAS MDDB JSP/Servlet (Version 2)*, and *All Models (Version 2)* palettes are maintained for backwards compatibility with AppDev Studio 2.0 projects and SAS Release 8.2 servers. In general, when creating new SAS AppDev Studio 3.0 projects that access SAS 9.1 servers, you should use the new components designed specifically for SAS 9.1 rather than the “Version 2” counterparts.