Paper 013-2011

Creating Custom Web Applications with SAS® 9.3 Stored Processes Using SAS® AppDev Studio™

Bruce Faulkner and Virgil Sealy, SAS Institute Inc., Cary, NC

ABSTRACT

This paper explains how to use SAS AppDev Studio to create custom Web applications that will run SAS stored processes on a SAS® 9.3 EBI server.

INTRODUCTION

SAS AppDev Studio, a set of plug-ins for the Eclipse IDE for Java EE Developers, helps you create customizable Java applications through a variety of templates that speed application development. Beginning with SAS AppDev Studio 3.4, a new set of templates helps you create Java client projects that run stored processes remotely. In SAS AppDev Studio 3.41, these generated applications execute SAS Stored Processes in a SAS 9.3 BI server or SAS 9.3 Enterprise BI server environment, and return results to the client.

Using SAS Stored Processes enables you to access the powerful capabilities of existing SAS programs. Many standard SAS products (such as the SAS® Stored Process Web Application, the SAS® Add-In for Microsoft Office, and SAS® Enterprise Guide) provide the ability to run stored processes. However, you might want to perform special processing for either the input or output of a stored process for better integration with your existing business logic.

The SAS Stored Process Service Java application programming interface (API) enables you to use Java to execute a stored process and retrieve the results. If you are not familiar with the SAS Stored Process Service Java API, the SAS AppDev Studio stored process templates help you by adding a simplifying abstraction for some of the API details. The templates guide you through creating custom Java clients, and these clients interact with a stored process running remotely on a SAS® Stored Process Server.

SAS AppDev Studio provides the ability to execute stored processes from either a custom Java console application or a Web application. SAS AppDev Studio generates code that lets you extend the application as needed to fit your specific requirements. This paper shows the details involved in creating a Web application that executes a SAS Stored Process.

SAS APPDEV STUDIO SUPPORT FOR GENERATING SAS STORED PROCESS CLIENTS

SAS AppDev Studio helps you create SAS Web Application Projects that use a servlet to execute a stored process and handle the results. The stored process servlet template produces a stored process driver servlet that uses the SAS Stored Process Service Java API for running a stored process. The generated servlet also processes the execution results. The servlet shows the results in your browser by default, but you can manipulate the results by extending the generated Java code. In addition, the servlet provides the ability to handle both replay and subsequent stored process execution.

To help you quickly build your custom Web application, SAS AppDev Studio generates one or more classes when you use the stored process servlet template.

The template always generates source code for the StoredProcessDriverServlet class. This class provides the overall logic for executing a stored process and manipulating the results. SAS AppDev Studio creates a separate stored process driver servlet for each specific stored process you want to execute. You can choose a name for the servlet when adding the template to a Web application project. Multiple driver servlets can exist in the same project, with each servlet running different stored processes.

SAS AppDev Studio produces two additional classes providing simplified access to the Stored Process Service Java API. These classes are:

- StoredProcessConnection, which manages the servlet's connection to SAS Foundation Services, including the Stored Process Service
- StoredProcessFacade, which encapsulates the service API and represents both a SAS Stored Process and its execution results (after the stored process executes)

The stored process template generates these two classes only once per Web application project. The StoredProcessConnection and StoredProcessFacade contain common code used by all the driver servlets in a project. The StoredProcessFacade provides the ability to access the StoredProcess2Interface and Execution2Interface so you can use methods on those interfaces if the StoredProcessFacade does not meet all your customization needs.

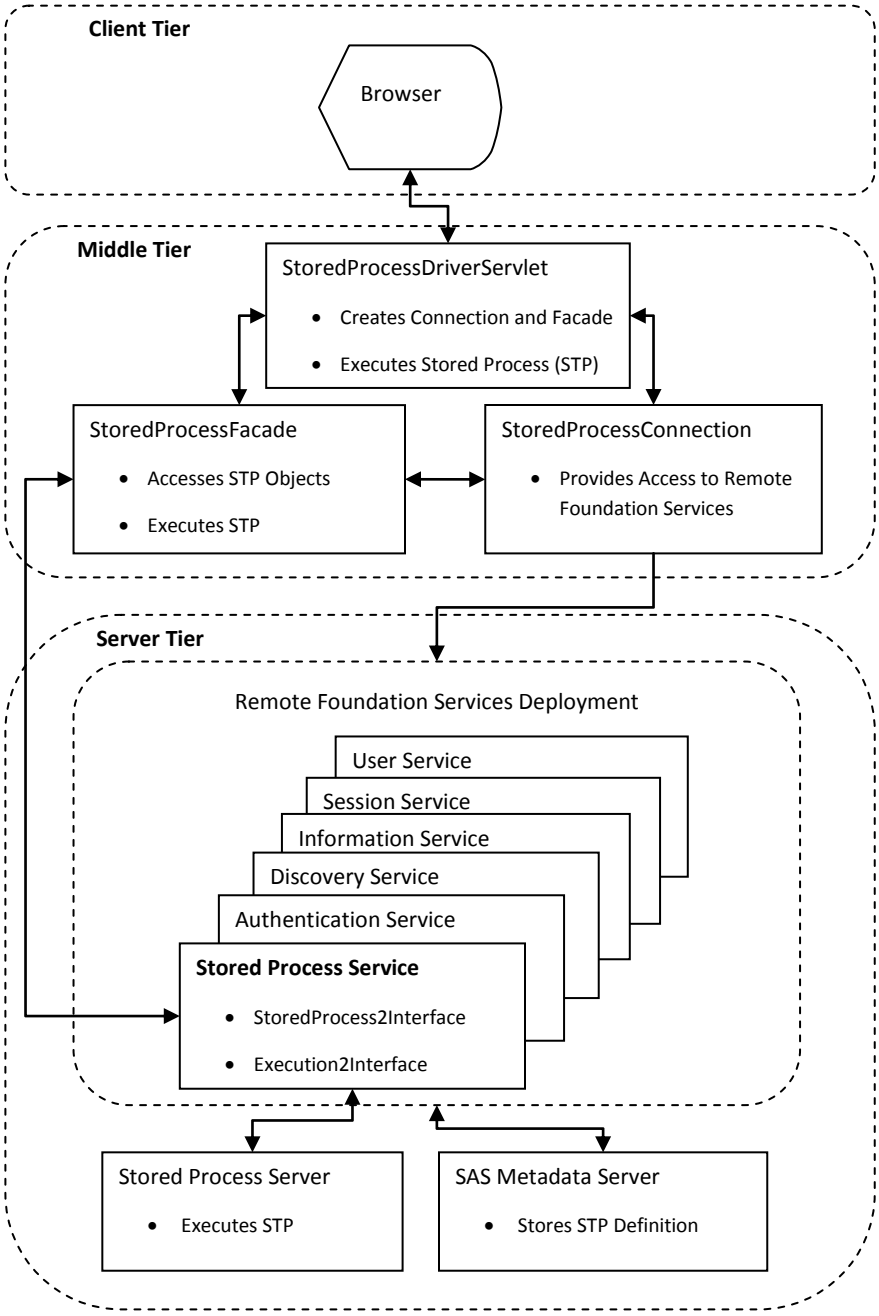

Figure 1. Overview of the Classes and Services Used by a SAS Web Application Project Containing a SAS Stored Process Servlet

As shown in Figure 1, the generated stored process driver servlet does the following:

- creates a StoredProcessConnection object
- creates a StoredProcessFacade object
- populates input parameters for the stored process
- executes the stored process
- writes the results to the browser
- provides methods to handle replay and subsequent stored process execution, if appropriate
- cleans up the StoredProcessFacade and StoredProcessConnection objects

You can extend the code in this servlet as needed, especially for setting input parameters and processing results. SAS AppDev Studio never regenerates the source code for the servlet, which preserves any changes you make to this class. Using this approach enables you to modify the servlet to meet your requirements, You will learn about the details of the generated code for the stored process driver servlet a little later in this paper.

The stored process servlet templates in SAS AppDev Studio support only the following two types of stored process results when executing a stored process on a SAS Stored Process Server:

- streaming results
- output parameter results

The stored process servlet template allows you to select stored processes that have either, neither, or both of these result types. As long as the stored process you select has at least one of the result types listed previously, SAS AppDev Studio will generate code for that stored process. If the stored process you select produces package results *in addition to* the result types listed, you access those results through the Execution2Interface exposed by the StoredProcessFacade. Currently, SAS AppDev Studio prevents completion of the stored process template wizard when you select a stored process that produces *only* package results.

In addition, SAS AppDev Studio does not generate specific code when the selected stored process has either of the following characteristics:

- data sources
- data targets

DEFINING OR MODIFYING A STORED PROCESS

A SAS Stored Process consists of a SAS program and specific metadata associated with that particular program. You can use either SAS Enterprise Guide or SAS® Management Console to create or modify a stored process. Both products enable you to define input parameters (prompts), output parameters, result types (streaming or package), and your SAS program.

Prior to SAS® 9.3, the source code for a SAS Stored Process exists on the file system. SAS 9.3 adds the ability to store the stored process source code in metadata. See the latest release of *SAS Stored Processes: Developer's Guide* for more information about stored processes.

BUILDING A SAS WEB INFRASTRUCTURE PLATFORM STORED PROCESS SERVLET

Starting with SAS 9.2, the SAS® Web Infrastructure Platform provides centralized access to various SAS services. The SAS Web Infrastructure Platform enables you to integrate a logon manager, which uses the SAS Authentication Service, into your Web application. In addition, the SAS Web Infrastructure Platform enables clients to access remote foundation services deployments instead of deploying local services. SAS AppDev Studio helps you generate Web applications that use the SAS Web Infrastructure Platform.

Use the following resources to learn more:

- See the SAS AppDev Studio developer's site at http://support.sas.com/rnd/appdev/. This site includes a PDF of the *SAS AppDev Studio Eclipse Plug-ins: User's Guide*. Both the site and the user's guide contain more details about the specifics of using SAS AppDev Studio.

- See the latest *SAS Intelligence Platform: Web Application Administration Guide* at http://support.sas.com/documentation/onlinedoc/intellplatform/index.html for specifics about the SAS Web Infrastructure Platform. (Starting with the release of SAS 9.3, information about the SAS Web Infrastructure Platform will be in a new document, *SAS Intelligence Platform: Middle-Tier Administration Guide*.)

SET UP YOUR WORKSPACE

You can easily build a Web application project that uses the SAS Web Infrastructure Platform and the Stored Process Service. First, follow the steps in the SAS AppDev Studio New Workspace Setup cheat sheet. This cheat sheet guides you through configuring your workspace so you can develop and run SAS Web application projects. (The cheat sheet displays automatically when you initially launch the SAS AppDev Studio Eclipse Plug-Ins with an empty Eclipse workspace.)

After you complete the cheat sheet, you might want to create a SAS Web Application Project to execute a stored process. To create the project, launch the New SAS Web Application Project wizard using one of these techniques:

- Launch the wizard from the Eclipse File->New Menu.
- Launch the wizard after selecting a SAS Stored Process in the tree displayed in the SAS Metadata Folders view in the SAS Metadata perspective.

CREATE A SAS WEB APPLICATION PROJECT WITH A STORED PROCESS

To use the SAS Metadata Folders view, first select the SAS Metadata perspective. Next, connect to a SAS Metadata Server by selecting the connection button in the Eclipse status bar. Then, select the SAS Metadata Folders view to show a tree containing the metadata objects found on that SAS Metadata Server. Expand the tree and select a stored process—this paper uses the **Sample: Multiple Output Formats** stored process. You can then use the context menu to create a new SAS Web Application Project as shown in Figure 2.

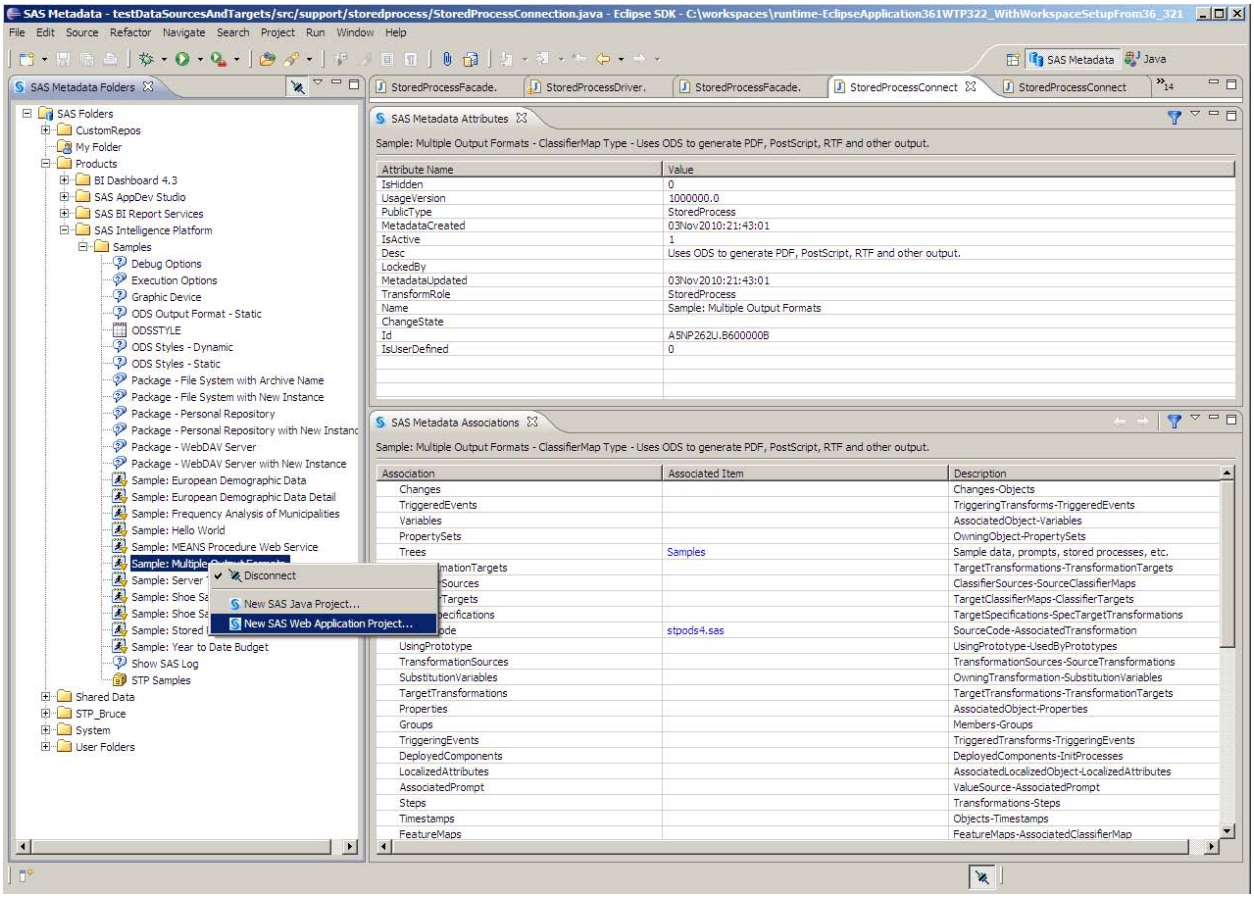

Figure 2. SAS Metadata Perspective after Selecting **New SAS Web Application Project**

SELECT DETAILS IN THE STORED PROCESS SERVLET TEMPLATE

After you select New SAS Web Application Project, SAS AppDev Studio launches the wizard to create your project. This wizard selects the SAS Stored Process Servlet Example template for you. Enter a project name in the wizard, such as **`SGF_2011_MultOutFormats`**. The remaining steps of the wizard guide you through creating the stored process servlet. The wizard enables you to set unique values for several other fields, including the example base name, the servlet class name, the servlet package, the servlet name, and the servlet URL pattern.

Several of the fields in the New SAS Web Application Project wizard have default values, including the following:

- Example base name: **`StoredProcessDriver`**
- Servlet class name: **`StoredProcessDriverServlet`**
- Servlet package: **`servlets`**
- Servlet name: **`StoredProcessDriverServlet`**

The wizard derives the default values for the servlet class name and the servlet name from the example base name.

The examples shown use **`SGF_2011_MultOutFormats`** as the project name. The other wizard fields use the default values. Of course, you might set unique values for these fields in your project.

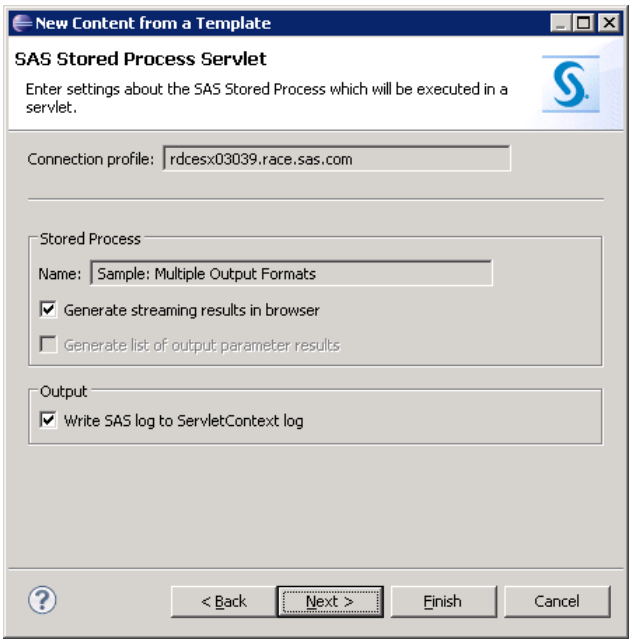

Figure 3. SAS Stored Process Servlet Page from the New SAS Web Application Project Wizard

After selecting defaults or entering values in the previous wizard pages, choose how you want to handle both the output results and the SAS log. This wizard contains additional pages, but the fields on those pages all have default values. If you want to accept the default values, select the Finish button to create your project. If you want to change or review the fields on subsequent wizard pages, select the Next button instead.

After you finish the wizard, a new Web application project appears in your workspace with the following three generated source code files:

- **StoredProcessDriverServlet.java** (in the **`servlets`** package)
- **StoredProcessConnection.java** (in the **`support.storedprocess`** package)
- **StoredProcessFacade.java** (in the **`support.storedprocess`** package)

Depending on your preference settings, Eclipse might ask you to switch to either the Java or Java EE perspectives. After you switch to one of these perspectives, open or change to the Eclipse Problems view. You might notice some errors in the Eclipse Problems view as shown in Figure 4. Before fixing any errors present, add two additional templates to the project. You need these templates before attempting to run the Web application.

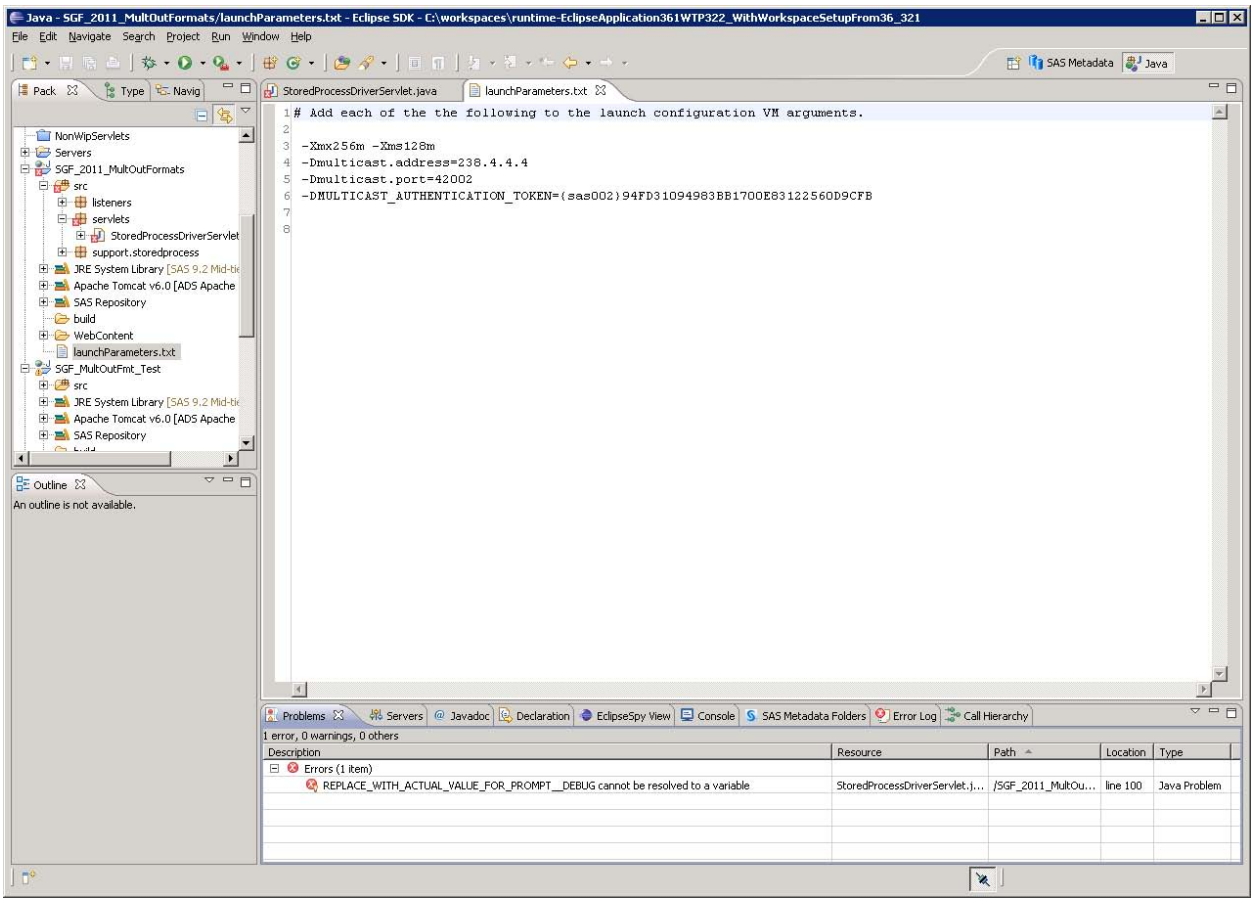

Figure 4. Eclipse Java Perspective with the **Problems View** Showing an Error in the Project

The problems listed in Figure 4 occur when SAS AppDev Studio cannot determine an appropriate value to generate for an input parameter value. As a result, SAS AppDev Studio generates an eye catcher to draw your attention to the values that you might provide. See the **USING INPUT PARAMETERS** section for more details about the eye catchers.

ADD THE EXAMPLES WELCOME PAGE TEMPLATE

First, add the Examples Welcome Page by selecting the following:

- File->New->Other->SAS AppDev Studio->Add Template Content to Project->SAS Web Infrastructure Platform Support->Examples Welcome Page

See Figure 5 for the results of selecting these items. The Examples Welcome Page template adds a JavaServer Pages (JSP) page to the Web application. When you run the Web application in SAS AppDev Studio, this JSP enables you to select any of the examples added to your project.

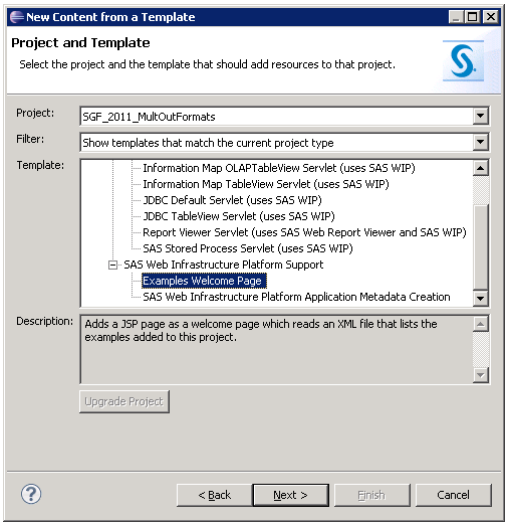

Figure 5. Add Template Content to Project Wizard Page with the **Examples Welcome Page** Template Selected

ADD THE APPLICATION METADATA CREATION TEMPLATE

Next, add support files for creating and deploying application metadata by selecting the following:

- File->New->Other->SAS AppDev Studio->Add Template Content to Project->SAS Web Infrastructure Platform Support->SAS Web Infrastructure Platform Application Metadata Creation

See Figure 6 for the results of this selection. This template creates several files for managing the SAS Web Infrastructure Platform metadata necessary for running your Web application. The filenames all begin with the application name you enter in this template. The application name defaults to the project name, but you can change this value.

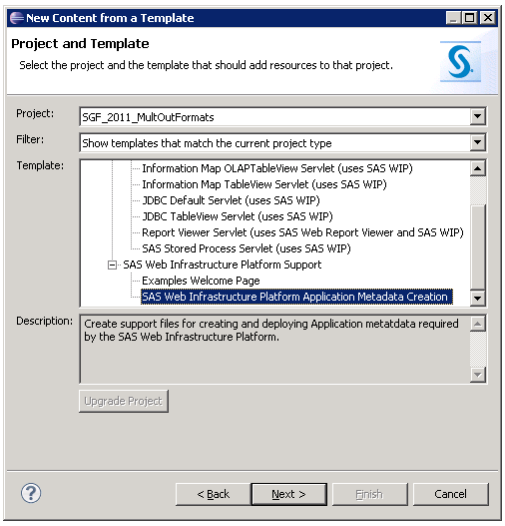

Figure 6. Add Template Content to Project Wizard Page with the **SAS Web Infrastructure Platform Application Metadata Creation** Template Selected

The Application Metadata Creation template uses the Web application name as the prefix in the filenames. Adding this template to your **SGF_2011_MultOutFormats** project produces a **SGF_2011_MultOutFormats Create Metadata.launch** file. After you finish the wizard for this template, run this launch file as shown in Figure 7. This launch file creates any necessary metadata to register your Web application on a SAS Metadata Server.

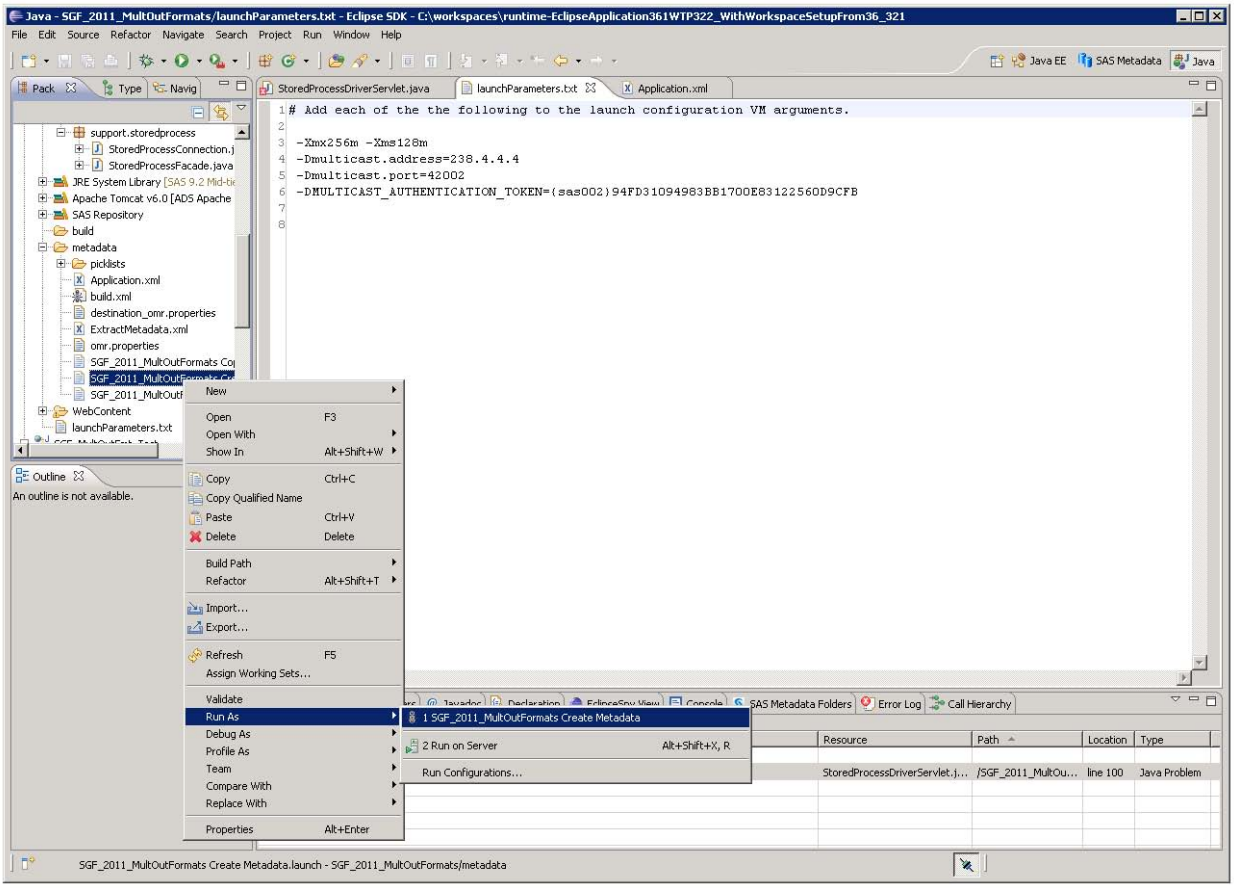

Figure 7. Eclipse Java Perspective with **Run As ->SGF_2011_MultOutFormats Create Metadata** Selected

SET INPUT PARAMETERS

Now you are almost ready to execute your Web application. First, correct any errors in the Eclipse Problems view. If SAS AppDev Studio generates an eye catcher value for one or more input parameters, you see errors in this view as shown in Figure 4.

Stored processes use prompts to define input parameters in metadata. An input parameter (or prompt) can have a default value. The generated driver servlet contains constant definitions for each prompt name and prompt value. SAS AppDev Studio generates a constant for the prompt value if the prompt (input parameter) data type has a default value and is one of the following types:

- `com.sas.datatypes.StringType`
- `com.sas.datatypes.IntegerType`
- `com.sas.datatypes.DoubleType`

If an input parameter has no default value or has a different data type from those listed above, then SAS AppDev Studio generates an eye catcher. You replace the eye catcher with an appropriate value for the input parameter as described below in the **USING INPUT PARAMETERS** section.

RUN THE WEB APPLICATION

After you replace the eye catcher values, run the Web application as shown in Figure 8. (See detailed instructions describing how to run Web applications in the *SAS AppDev Studio Eclipse Plug-ins: User's Guide*, available from the SAS AppDev Studio developer's site at http://support.sas.com/rnd/appdev/.)

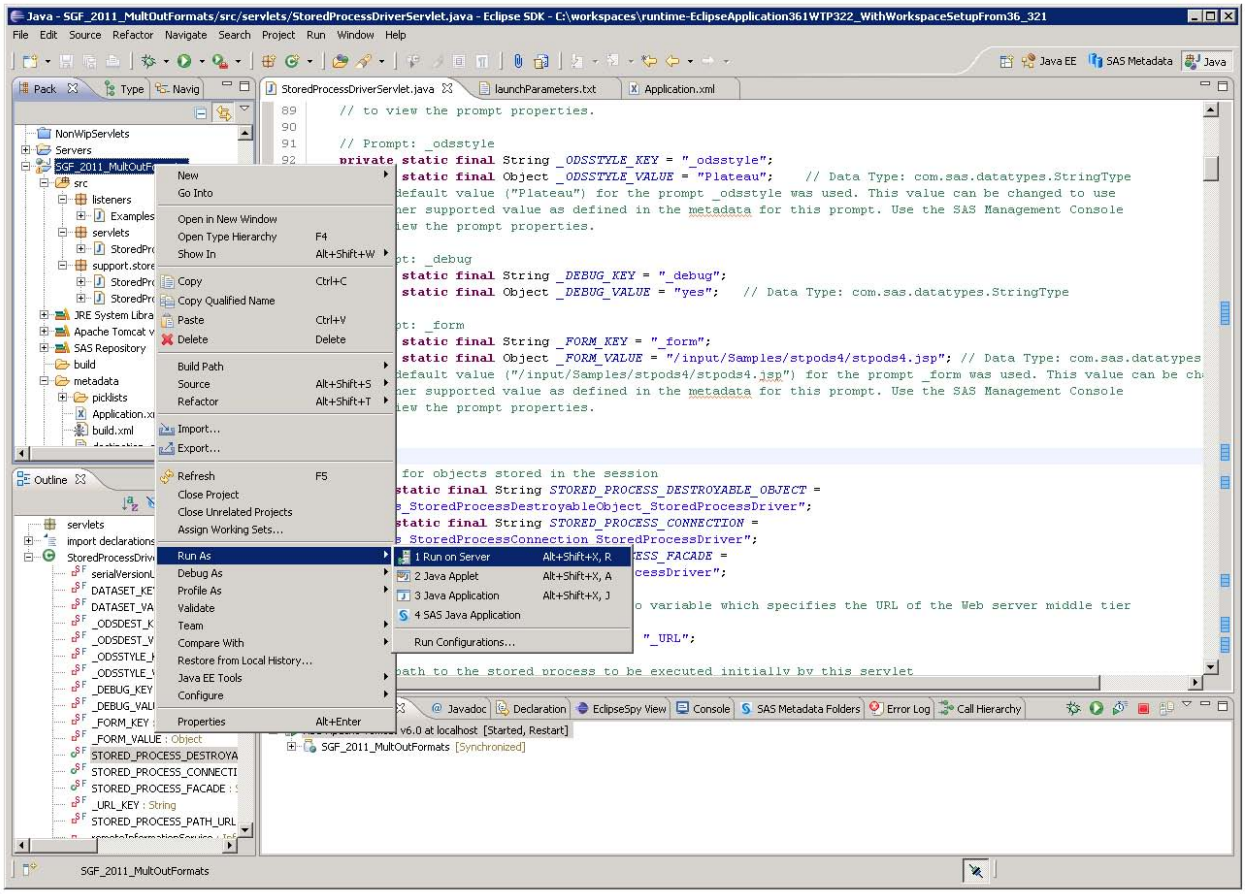

Figure 8. Eclipse Java Perspective with the Project Name Selected, Followed by **Run As**, and then **Run on Server**, to Run the Web Application

When you run your application from SAS AppDev Studio, a browser window opens to the logon page as shown in Figure 9.

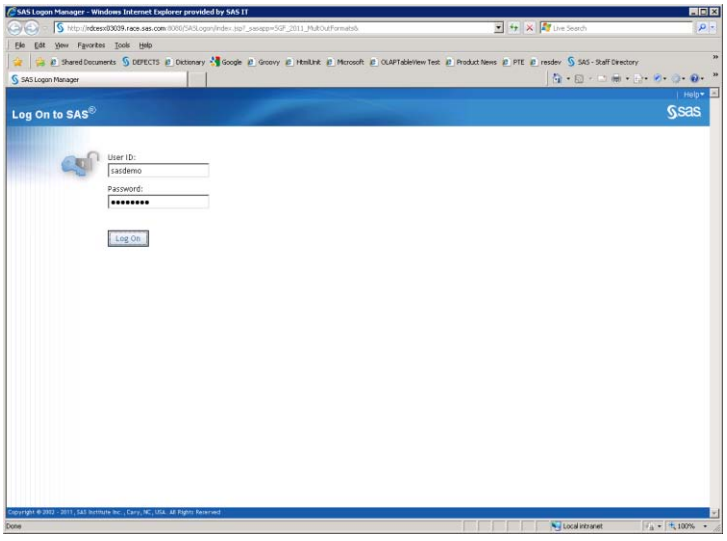

Figure 9. Initial SAS Logon Screen for the Web Application

After a successful logon to the Web application, you see the list of all example templates you added to this project. To execute the stored process and see the results, select the StoredProcessDriverServlet example entry you just created, as seen in Figure 10.

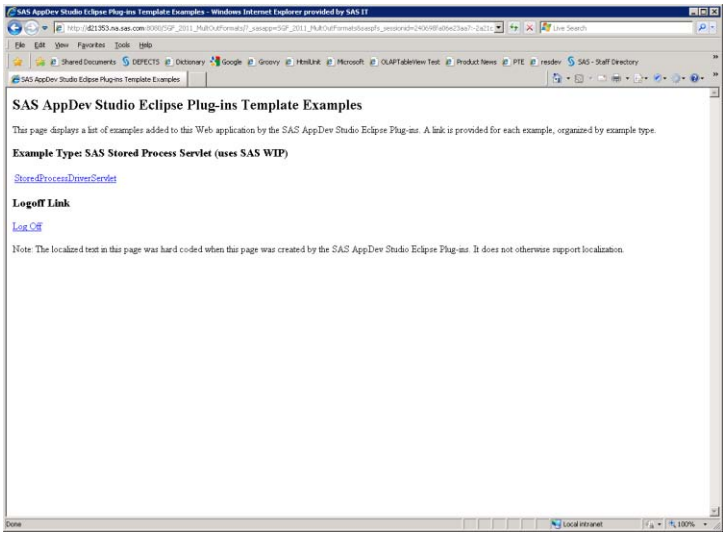

Figure 10. SAS AppDev Studio Examples Welcome Page

After you select the StoredProcessDriverServlet link, the stored process driver servlet executes the **Sample: Multiple Output Formats** stored process on the SAS Stored Process Server. This sample stored process uses the Output Delivery System (ODS) to return output in different forms. By default, this sample stored process produces HTML results as you see in Figure 11.

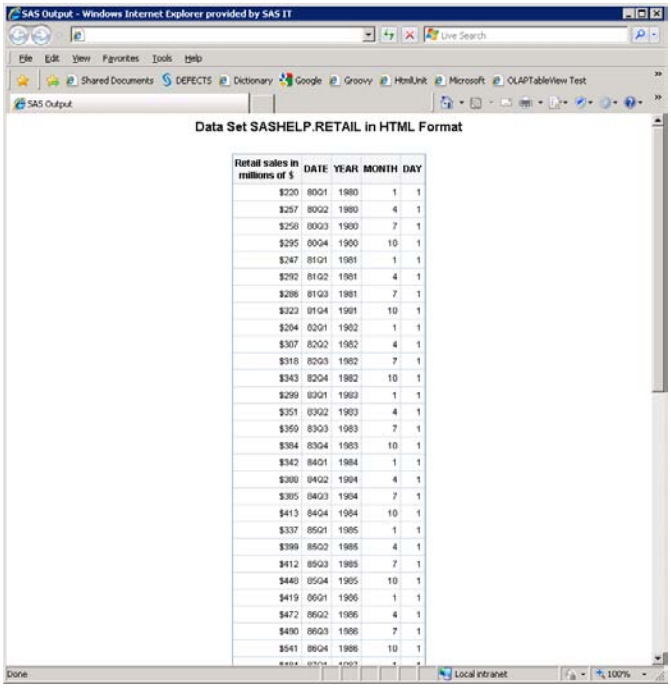

Figure 11. Browser Showing the HTML Results after Executing the Stored Process Driver Servlet

In addition to creating HTML results, the **Sample: Multiple Output Formats** stored process can produce other output types. This stored process uses the **`_odsdest`** input parameter to select the results format. Use SAS Management Console to view the values for this prompt, as shown below in Figure 12.

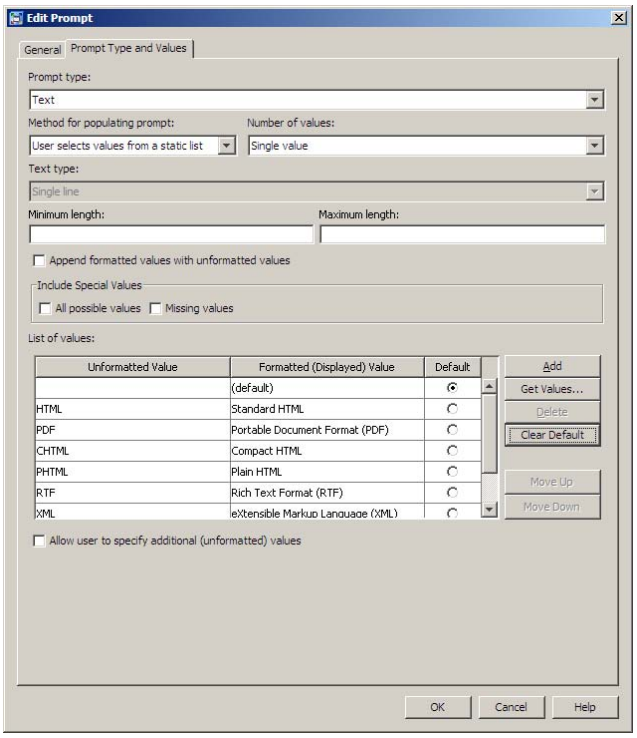

Figure 12. Viewing the Values for the **`_odsdest`** Input Parameter (Prompt) of the **Sample: Multiple Output Formats** Stored Process in the SAS Management Console

In your StoredProcessDriverServlet, you might set the value of the `_odsdest` input parameter to one of the values listed for the prompt. Modify the value of the `_ODSDEST_VALUE` constant to match an unformatted value listed in the SAS Management Console for this stored process, such as PDF. See Figure 12 for an example showing the unformatted values. After you publish and run the Web application again, you see the PDF results in Figure 13.

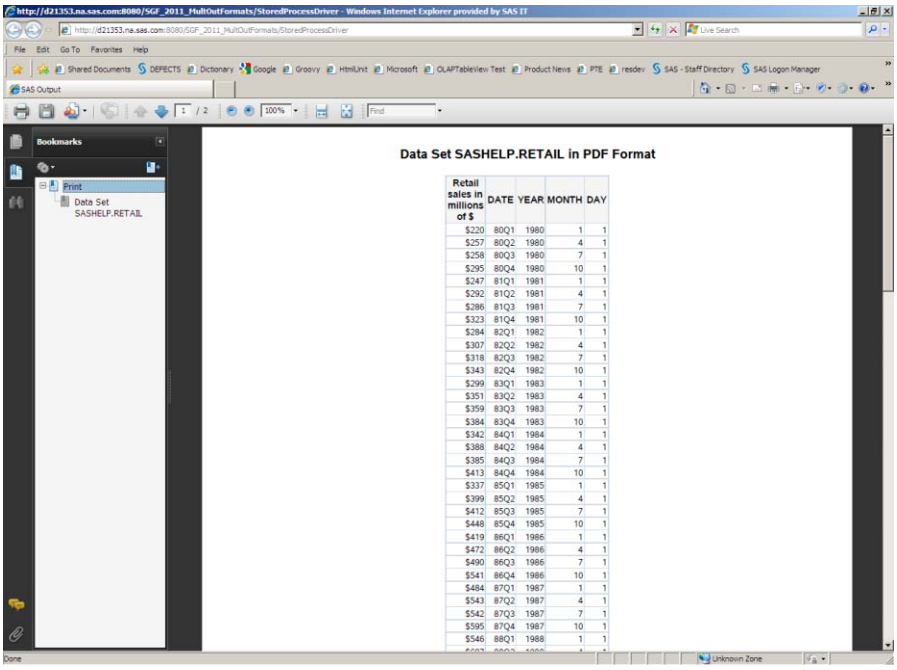

Figure 13. Browser Showing the PDF Results after Executing the Stored Process Driver Servlet

The **Sample: Multiple Output Formats** stored process has other input parameters. Changing the values for these parameters changes the results or their appearance. For example, you might change the data set used by the stored process or the ODS style for the results. Again, use the SAS Management Console to determine appropriate values for the input parameters. Once you know the appropriate values, change the constant definitions in the StoredProcessDriverServlet. Publish and run the Web application again to see the changes.

CONTENTS OF THE GENERATED STORED PROCESS DRIVER SERVLET

Now that you know how to create a stored process driver servlet, you might want to customize the servlet. Before modifying this code, you should learn the contents of the generated servlet source code, which provides a base for running a stored process and making the results available in the browser. Understanding the servlet source code helps when you change the code to perform custom processing for your Web application.

IDENTIFYING THE STORED PROCESS PATH URL

Metadata defines the characteristics of a SAS Stored Process, including the location of the stored process in the metadata repository. A SAS Business Intelligence Protocol (SBIP) Uniform Resource Locator (URL) describes the stored process location in the repository. After you select a SAS Stored Process in the template wizard, SAS AppDev Studio generates a constant for the SBIP URL.

The generated constant appears in the servlet like the following snippet:

```
// The path to the stored process to be executed initially by this servlet
private static final String STORED_PROCESS_PATH_URL =
    "SBIP://METASERVER/Products/SAS Intelligence Platform/Samples/" +
    "Sample: Multiple Output Formats(StoredProcess)";
```

The generated string contains the SBIP URL completely on one line, but this snippet splits the string for clarity.

It is not advisable to change the SBIP URL after generating the servlet. The driver servlet contains other generated code reflecting capabilities of that stored process as well as choices you made in the wizard. If you want to change the stored process initially executed by the servlet, use the SAS AppDev Studio templates to add another SAS Stored Process Servlet to your project. When you create the new servlet, select the new stored process to execute, and give the driver servlet a different name. Creating a new driver servlet ensures that the generated code handles the characteristics of the stored process.

CREATING HELPER OBJECTS

The StoredProcessDriverServlet creates instances of the StoredProcessConnection and StoredProcessFacade objects. These helper objects provide simplified access to several services used for executing a stored process and for handling the results in the driver servlet.

The StoredProcessConnection provides convenient access to SAS Foundation Services, a SAS Metadata Server, and the Stored Process Service. The StoredProcessConnection also surfaces methods to search for stored process metadata objects in a SAS Metadata repository.

The Stored Process Service Java API provides interfaces for working with SAS Stored Processes. One of the Java interfaces, com.sas.services.storedprocess.StoredProcess2Interface, represents a SAS Stored Process, and the com.sas.services.storedprocess.Execution2Interface provides access to the execution results.

The StoredProcessFacade encompasses a SAS Stored Process along with its execution results—of course, the results only become available after the stored process runs. The StoredProcessFacade wraps access to two main elements of the Stored Process Service Java API: the StoredProcess2Interface and the Execution2Interface. By providing access to these interfaces through simplifying abstractions, the StoredProcessFacade hides the full details of the implementation of those interfaces.

Some of the key helper methods in the StoredProcessFacade handle:

- setting input parameters and other attributes of a stored process
- processing stored process streaming and output parameter results
- dealing with the content type and character encoding of streaming results
- retrieving from and writing to the SAS log

- creating and executing replay of stored process objects
- retrieving the underlying instances of StoredProcess2Interface and Execution2Interface objects

As mentioned earlier in the **SAS APPDEV STUDIO SUPPORT FOR GENERATING SAS STORED PROCESS CLIENTS** section, SAS AppDev Studio generates the source code for the StoredProcessConnection and StoredProcessFacade only once per Web application project. The generated source code resides in the **`support.storedprocess`** package. Do not modify the generated source code for these two classes. Each stored process driver servlet creates instances of these classes. The servlet uses these instances to execute a SAS Stored Process and process its results.

The driver servlet also creates a StoredProcessDestroyableObject. The servlet adds this object to both the ExamplesSessionBindingListener and the HttpSession. Once the servlet creates and initializes the StoredProcessConnection and StoredProcessFacade, the servlet then places these common objects into the StoredProcessDestroyableObject. This object cleans up the StoredProcessConnection and StoredProcessFacade when the HTTP session expires after running the servlet.

USING INPUT PARAMETERS

The StoredProcessDriverServlet uses an instance of a StoredProcessFacade to gain simplified access to the SAS Stored Process Service Java API, so you can focus on developing custom logic in the Java servlet, if necessary. When the StoredProcessDriverServlet creates a StoredProcessFacade instance, the servlet passes a `java.util.Map` object to the StoredProcessFacade constructor. This map contains the input parameters for the stored process.

What Are Input Parameters?

Input parameters to a stored process are optional. If a stored process contains input parameters, then the StoredProcessFacade constructor accepts values for those input parameters to control stored process execution. Input parameters have a name and a value; they exist in SAS programs as global macro variables. You define input parameters (also known as prompts) in the stored process metadata using the SAS Management Console or SAS Enterprise Guide. The StoredProcess2Interface establishes the input parameter *name* as a `java.lang.String` and the *value* as a `java.lang.Object`.

How Does SAS AppDev Studio Generate Code For Input Parameters?

The StoredProcess2Interface enables you to supply input parameters to a stored process via a map object, with each entry in the map having a key and a value. When you select a stored process in a SAS AppDev Studio stored process template, the wizard retrieves the list of input parameters (if any) from the metadata. In the code that SAS AppDev Studio generates, the input parameter *name* becomes the *key* in the map entry, while the input parameter *value* becomes the *value* of the map entry.

As described above in the **SET INPUT PARAMETERS** section, when certain types of input parameters (`com.sas.datatypes.StringType`, `com.sas.datatypes.IntegerType`, and `com.sas.datatypes.DoubleType`) have a default value in metadata, then the generated code contains a constant for the map value using the actual prompt default value.

For all other types of input parameters or for any of the types listed above without a default value defined, the generated constant for the map value has the text **`REPLACE_WITH_ACTUAL_VALUE_FOR_PROMPT_YYY`**, where **`YYY`** is the actual name of the input parameter. The next section shows examples of these constants.

SAS AppDev Studio creates these constants for centralized access and readability in the servlet, but you might want to set input parameters programmatically, depending on the logic required for your application. The examples in this paper use the generated constant values for simplicity and clarity.

The **`populateParameterMap()`** method of the StoredProcessDriverServlet uses the automatically generated constants to add input parameter entries to a map. The servlet supplies the input parameter `Map` to the StoredProcessFacade, which provides this map to the Stored Process Service Java API.

What Code Do You Modify To Override The Generated Input Parameter Values?

Looking at some sample code clarifies how the code that was generated by SAS AppDev Studio uses input parameters. This example assumes that you have defined a stored process in your metadata repository with the name **`addintegers`**. This stored process takes two integer input parameters and returns the sum as an output parameter. The stored process uses the following SAS program as its source code:

```
/* addintegers.sas
 *
 * Accepts two integer input parameters: int1 and int2, adds them, and returns
 * the result as an output parameter.
 */

%global int1 int2;

*ProcessBody;

%let Sum = %eval(&int1 + &int2);
%put Adding of &INT1 + &INT2 yields: &Sum;
run;
```

The metadata for **addintegers** contains the definitions of these input parameters as numeric prompts. The names of these input parameters are **int1** and **int2**. These input parameters do not have default values defined.

A snippet of the generated Java code in the driver servlet shows the constant definitions for the input parameters:

```
// Prompt: int1
private static final String INT1_KEY = "int1";
// Data Type: com.sas.datatypes.IntegerType
private static final Object INT1_VALUE = REPLACE_WITH_ACTUAL_VALUE_FOR_PROMPT_INT1;

// Prompt: int2
private static final String INT2_KEY = "int2";
// Data Type: com.sas.datatypes.IntegerType
private static final Object INT2_VALUE = REPLACE_WITH_ACTUAL_VALUE_FOR_PROMPT_INT2;
```

Because no definitions exist for the eye catchers, you see an error for each eye catcher in the Eclipse Problems view. Before compiling the driver servlet, supply appropriate values for the eye catchers.

Say you want to add 2 + 3 in your servlet. You modify the eye catchers using the actual values, which results in the following snippet of the driver servlet:

```
// Prompt: int1
private static final String INT1_KEY = "int1";
// Data Type: com.sas.datatypes.IntegerType
private static final Object INT1_VALUE = 2;

// Prompt: int2
private static final String INT2_KEY = "int2";
// Data Type: com.sas.datatypes.IntegerType
private static final Object INT2_VALUE = 3;
```

The **populateParameterMap()** method in the driver servlet uses these constants, so you do not have to change this method for any parameters with generated constants. This method automatically builds a map of the input parameters using these constants. The driver servlet calls **populateParameterMap()** and supplies the resulting parameter map to the StoredProcessFacade constructor.

In the following example, you see how the generated **populateParameterMap()** method appears for the driver servlet using the `addintegers` stored process:

```
/**
 * Place the keys and values for the stored process prompts (input parameters) into
 * a map. The map can then be provided to the StoredProcessFacade to supply the
 * input parameters to the SAS Stored Process.
 */
private Map<String, Object> populateParameterMap()
{
    final Map<String, Object> storedProcessParameters =
           new HashMap<String, Object>();

    // Set the prompt (input parameter) values
    storedProcessParameters.put(INT1_KEY, INT1_VALUE);
```

```
        storedProcessParameters.put(INT2_KEY, INT2_VALUE);

        return storedProcessParameters;
    }
```

Stored processes that have input parameters defined in metadata can generate one or more global macro variables for each input parameter. However, you might have stored processes containing global macro variables not associated with input parameters. The stored process metadata contains no information for global macro variables without associated prompt definitions. As a result, SAS AppDev Studio cannot generate constants for global macro variables in the stored process driver servlet.

You might have a stored process that uses global or reserved macro variables. Because the stored process metadata contains no definitions for these macro variables, you modify the generated **`populateParameterMap()`** method to supply values for these macro variables.

For example, the SAS code for the **Sample: European Demographic Data** stored process uses a reserved macro variable, **`_GRAFLOC`**, to indicate the location of the associated SAS Graph applets and ActiveX control. The **`_GRAFLOC`** variable must contain an appropriate path for the resulting browser client to work properly. To set this value, add the following highlighted code to the **`populateParameterMap()`** method:

```
    /**
     * Place the keys and values for the stored process prompts (input parameters) into
     * a map. The map can then be provided to the StoredProcessFacade to supply the
     * input parameters to the SAS Stored Process.
     */
    private Map<String, Object> populateParameterMap()
    {
        final Map<String, Object> storedProcessParameters =
               new HashMap<String, Object>();

        // Set the prompt (input parameter) values
        storedProcessParameters.put(INT1_KEY, INT1_VALUE);
        storedProcessParameters.put(INT2_KEY, INT2_VALUE);

        // Set _GRAFLOC to access SAS Graph Java applet or ActiveX control
        storedProcessParameters.put("_GRAFLOC", "/sasweb/graph");

        return storedProcessParameters;
    }
```

Of course, you might want to use constants for any other macro variable values instead of using hardcoded parameter values.

EXECUTING THE STORED PROCESS

After creating the helper objects and setting appropriate parameters and attributes for the stored process, the driver servlet calls the `executeStoredProcess()` method of the StoredProcessFacade. This method provides the input parameters to the underlying StoredProcess2Interface object before invoking the `execute()` method on that object. The `executeStoredProcess()` method returns the results from executing the stored process.

PROCESSING THE STORED PROCESS EXECUTION RESULTS

Depending on the type of results the SAS Stored Process produces and on the result types you select in the SAS Stored Process servlet wizard, the generated driver servlet contains code to handle output parameter results, streaming results, both, or neither.

Processing Output Parameters

The `writeOutputParameterList()` method of the StoredProcessDriverServlet simply writes some basic HTML into the HttpServletResponse for the browser to show. Each output parameter value appears on a new line in the HTML. This method also writes each output parameter value to the log of the servlet context. You might want to override this method to process the output parameters in some other way, either depending on the SAS Stored Process you execute or on other requirements of your Web application.

Displaying Streaming Results

For stored processes that produce streaming results, the StoredProcessDriverServlet writes those results to the OutputStream of the HttpServletResponse. SAS Stored Processes returning streaming results typically use ODS to produce those results. The SAS Integration Technologies **Sample: European Demographic Data** stored process shows how a stored process uses ODS to write streaming results back to the servlet. These results then appear in your browser. Similar to output parameter processing, you might override the `writeStreamingResults()` method to process the results differently than the supplied method.

Showing the SAS Log

While developing your project in SAS AppDev Studio, you might need to view the SAS log to help debug any problems—not that you ever have any of those! When you add a stored process template to your project, the SAS AppDev Studio wizard enables you to select a check box for displaying the contents of the SAS log in the servlet context log.

Even though the check box controls whether logging is active, SAS AppDev Studio always generates code for writing to the log. However, the driver servlet uses the **`showSASLog`** Java variable for controlling whether the servlet writes the output to the servlet context log. The value of **`showSASLog`** initially comes from the selection you made when adding the template content to the project, but you might change the value to show the log as needed during project development.

EXECUTING A REPLAY STORED PROCESS

Along with support in the generated StoredProcessFacade, the StoredProcessDriverServlet provides the ability to execute a replay stored process. The replay stored process feature uses a SAS Stored Process Server session to execute an additional in-memory stored process to provide more results to a client.

SAS Integration Technologies provides some sample stored processes that provide insight into how you might use various SAS Stored Process characteristics. For example, the SAS Integration Technologies **Sample: European Demographic Data** stored process uses a replay stored process to produce the clickable map of Europe in the ODS results.

The replay stored process uses the *same* StoredProcessFacade object as the original stored process.

EXECUTING A SUBSEQUENT STORED PROCESS

The StoredProcessDriverServlet also provides the ability to execute a subsequent stored process after completing the initial stored process (and perhaps after completing a replay stored process). The subsequent stored process can also have a replay stored process.

Once again, the SAS Integration Technologies **Sample: European Demographic Data** stored process provides an example of a subsequent stored process. When you select a country in the map displayed (via replay) after running the original stored process, the **Sample: European Demographic Data Detail** stored process executes subsequently. This stored process produces ODS results for the detailed data for the country you selected.

The StoredProcessDriverServlet creates a *new* StoredProcessFacade object to execute a subsequent SAS Stored Process.

CUSTOMIZING THE GENERATED SAS STORED PROCESS SERVLET

Now that you have seen the generated code resulting from adding a SAS Stored Process Servlet template to a SAS AppDev Studio Web Application project, you might wonder how you would customize the generated servlet to perform additional processing in Java. Throughout this paper, you have seen various explanations of customizations you can make.

Here is a list of the customizations presented throughout the paper, along with some other ideas for additional processing you might consider:

- programmatically set input parameters in the servlet
- create a custom user interface to let your users supply input parameter values
- programmatically set additional global macro variables for the stored process
- override the `writeOutputParametersList()` method to get the output parameter values and process them before displaying them
- override the `writeStreamingResults()` method to get the results and process them before displaying them
- retrieve the Execution2Interface object from the StoredProcessFacade to access package results
- override the `executeSubsequentStoredProcess()` method to retrieve values from the HTTP session object or to set additional stored process options
- override the `handleExecutionStatus()` method to add logic based on the completion status or SAS return code
- override the `writeSASLog()` method to process the log or write log results somewhere other than the ServletContext log

You can create additional Java code to execute in your application as well.

For more information about SAS Stored Processes, the SAS Stored Process Service Java API, and other related concepts, see the following:

- SAS AppDev Studio developer's site
- *SAS AppDev Studio Eclipse Plug-ins: User's Guide*
- *SAS Stored Processes: Developer's Guide*
- *SAS Intelligence Platform: Web Application Administration Guide* (starting with the SAS 9.3 release, see *SAS Intelligence Platform: Middle-Tier Administration Guide*)

These references give you more detail to help you customize the generated code.

In addition, see the package Javadoc for the `com.sas.services.storedprocess` package. To find the most recent Javadoc:

1. Navigate to the SAS AppDev Studio developer's site at http://support.sas.com/rnd/appdev/.
2. Select the *SAS BI API Documentation* link.
3. Select the *SAS Foundation Services* link from the SAS BI Packages Libraries.
4. Select the `com.sas.services.storedprocess` package.
5. Scroll down in the package Javadoc to read detailed information about working with the Stored Process Service Java API.

CONCLUSION

SAS AppDev Studio helps you develop custom Web applications that use the power of SAS Stored Processes. You have seen that SAS AppDev Studio generates source code as a foundation for working with stored processes. You have the ability to modify the generated servlet to implement different or additional business logic in Java.

You learned how to do the following:

- use SAS AppDev Studio features for generating code executing SAS Stored Processes
- build a SAS Web Infrastructure Platform Stored Process servlet from either the File->New menu or from the SAS Metadata Folders view
- understand the contents of the generated StoredProcessDriverServlet, StoredProcessConnection, and StoredProcessFacade and how to use them
- take advantage of input and output parameters
- register your Web application on a SAS Metadata Server

- customize the generated code

You create customizable stored process Web applications through the templates that SAS AppDev Studio supplies. These templates help you easily build the foundation code for servlets running stored process servlets. The generated StoredProcessConnection and StoredProcessFacade objects provide a layer of abstraction, which avoids repeating code unnecessarily. You might even modify the StoredProcessDriverServlet to tailor the result handling for your application requirements.

ACKNOWLEDGMENTS

We acknowledge and greatly appreciate the help on this paper from several of our colleagues at SAS Institute Inc. Our colleagues made a careful review this paper, and they provided numerous helpful suggestions and corrections that improved this paper. We thank the following colleagues:

- Denise Crosson validated the accuracy of the technical content of this paper. As always, Denise paid careful attention to detail and goes the extra step in ensuring high quality.
- Gordon Hirsch provided excellent suggestions about the structure and content of the paper. His insight especially provided focus on explaining new concepts to readers of this paper.
- Rich Main articulated the original vision for the addition of stored process templates to SAS AppDev Studio. He provided much appreciated support during the development of the templates, and he provided several suggestions for improving this paper.
- Ernest Pasour gave valuable feedback on the paper. He always displayed keen technical insight during the development of the SAS AppDev Studio stored process templates.

RECOMMENDED READING

- SAS AppDev Studio developer's site at http://support.sas.com/rnd/appdev/
- SAS AppDev Studio Eclipse Plug-ins: User's Guide (located on the SAS AppDev Studio developer's site)
- SAS Stored Processes: Developer's Guide
- SAS Intelligence Platform: Web Application Administration Guide
- SAS Intelligence Platform: Middle-Tier Administration Guide (available with the SAS 9.3 release)
- SAS Intelligence Platform: System Administration Guide
- SAS Intelligence Platform: Overview
- SAS Integration Technologies: Overview
- SAS Integration Technologies: Java Client Developer's Guide
- SAS Foundation Services: Administrator's Guide
- SAS Intelligence Platform: Application Server Administration Guide

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Bruce Faulkner
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Work Phone: +1 919 531 6846
Fax: +1 919 677 4444
E-mail: Bruce.Faulkner@sas.com
www.sas.com

Virgil Sealy
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Work Phone: +1 919 531 5182
Fax: +1 919 677 4444
E-mail:Virgil.Sealy@sas.com
www.sas.com