THE POWER TO KNOW.

SAS® 9.2 BI Web Services Developer's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2009. *SAS® 9.2 BI Web Services: Developer's Guide*. Cary, NC: SAS Institute Inc.

SAS® 9.2 BI Web Services: Developer's Guide

ISBN 978-1-59994-835-5

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

1st electronic printing, February 2009

1st printing, February 2009

Contents

What’s New

Overview

SAS 9.2 BI Web Services provide generated Web services, as well as general enhancements.

Generated Web Services

SAS BI Web Services now provides two services: the XMLA Web services (as provided in SAS 9.1) and the new generated Web services. Generated Web services enable you to define and create your own customized Web services. The following new features are available with generated Web services:

- The Deploy As Web Service wizard enables you to select a set of stored processes in SAS Management Console and deploy them as a generated Web service.
- The Web Service Maker is a Web service that generates new Web services from stored processes.
- Generated Web services have a unique WSDL for the stored processes that are used in that Web service. This feature makes generated Web services easier for clients to call than the XMLA Web services, which use a generic, fixed WSDL for all stored processes.
- More output options are supported. XMLA Web services support only a single, inline XML output stream. Generated Web services support an inline XML output stream, multiple output streams (which can have XML or binary content), and output parameters. Binary content uses Web service attachments.
- Attachments (binary or text) are supported. Attachments enable you to send non-xml (binary) data with a SOAP request or response.
- Contract-first programming is supported by associating schemas with input and output streams. The schemas can be included in the generated WSDL. Contract-first is a technique that enables organizations to standardize on the XML formats they exchange between services. In SAS, the XML libname engine and XML maps are used to interpret the XML described by these schemas.
- Generated Web services are represented as objects in SAS Management Console, and they can be exported and imported by using the promotion wizards.

General Enhancements

The following general enhancements have been added to SAS BI Web Services:

- XMLA Web services have been enhanced to use the new prompting framework.
- JBoss and IBM WebSphere are now supported (in addition to BEA WebLogic) J2EE Application Servers. Apache Tomcat is no longer supported.

CHAPTER 1 Overview of SAS BI Web Services

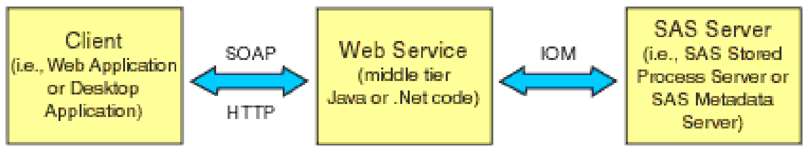
What Are SAS BI Web Services?

A Web service is an interface that enables communication between distributed applications. Web services enable cross-platform integration by enabling applications that are written in various programming languages to communicate by using a standard Web-based protocol, typically the Simple Object Access Protocol (SOAP). This functionality makes it possible for businesses to bridge the gaps between different applications and systems.

There are two implementations of SAS BI Web Services: one written in Java that requires a servlet container, and another written in C# that uses the .NET framework. For information about the differences between SAS BI Web Services for .NET and SAS BI Web Services for Java, see "Deciding Between .NET and Java" on page 2.

The following figure shows how Web services work.

Figure 1.1 Web Services Communications


Client (i.e., Web Application or Desktop Application)
SOAP
HTTP
Web Service (middle tier Java or .Net code)
IOM
SAS Server (i.e., SAS Stored Process Server or SAS Metadata Server)

A client, such as a Web application or desktop application, obtains the Web Service Description Language (WSDL) from the Web service. The WSDL describes the methods that are available, the endpoint (where to call the Web service), and the format of the XML that is required to call the Web service.

Web service clients and servers transport XML data by using a data envelope that is called a SOAP envelope. Any client that can send and receive SOAP messages can access Web services. SAS supports SOAP bindings over HTTP. The client sends XML requests and parameters in a SOAP envelope to the Web service, which tells the Web service to either discover or execute stored processes.

Discover and Execute are the two methods that are specified for XMLA. The Discover method consists of middle-tier code that calls the SAS Metadata Server to get the requested metadata. The Execute method consists of middle-tier code that calls the SAS Stored Process Server to invoke stored processes.

The stored process that is requested is executed by a SAS Stored Process Server. Usually any number of simple string parameters are passed to the stored process, and a stream of XML data is returned. The input parameters to the stored process can be nothing or a combination of simple string parameters and any number of XML streams.

Before SAS 9.2, you could write XML for Analysis (XMLA) Web services only. Starting with SAS 9.2, you can use SAS Management Console to deploy a set of stored processes as a generated Web service. For more information, see “What Are Generated Web Services?” on page 23.

Deciding Between .NET and Java

Installation and Administration Differences

- *What operating environment are you using?*
  - SAS BI Web Services for .NET can be installed only in the following operating environments:
    - Windows XP
    - Windows 2003
    - Windows Vista Business
    - Windows Vista Ultimate
    - Windows Vista Enterprise
    - Windows Server 2008
  - SAS BI Web Services for Java can be installed in any operating environment that is supported by SAS middle tier components.
- *How do you want to administer the Web services?*
  - SAS BI Web Services for .NET are administered by using IIS.
  - SAS BI Web Services for Java are administered by using JBoss, BEA WebLogic, or IBM WebSphere. If you install SAS BI Web Services for Java, then you also need to have a Java Virtual Machine for an application server.
- *Which Web service stack do you want to use?* (A stack is the engine that performs SOAP processing.)
  - SAS BI Web Services for .NET use the .NET Web service stack (Web Services Enhancements 3.0). The .NET stack supports Message Transmission Optimization Mechanism (MTOM) attachments.
  - SAS BI Web Services for Java use the Apache Axis2 Web service stack. This stack supports both MTOM attachments and SOAP Messages with Attachments (SwA). If your service uses attachments and your client is .NET, then use MTOM, which is enabled by default.

SAS BI Web Services for .NET and SAS BI Web Services for Java also differ in the way they handle logging and configuration. SAS BI Web Services for .NET use .NET logging, and SAS BI Web Services for Java use SAS Foundation Services for logging. For more information about configuring Web services, see the *SAS Intelligence Platform: Web Application Administration Guide*.

Client Differences

Web service clients cannot generally identify differences between SAS BI Web Services for .NET and SAS BI Web Services for Java. If the client conforms to the usage rules for Web services, then the client should be able to use either platform.

One difference for the Web service client is how the path is returned for the name of the stored process (DataSourceName). SAS BI Web Services for Java return a fully qualified path. SAS BI Web Services for .NET return a simple path. In both cases, you get the name of the stored process to invoke by using the Discover method of the same Web service.

Creating SAS BI Web Services

Use Web Services: Prerequisite

Before you can use Web services, you need to perform the following steps:

1 Decide whether you want to use SAS BI Web Services for .NET or SAS BI Web Services for Java. Install SAS BI Web Services and the SAS Metadata Server.

   Note: When you install SAS 9.2 BI Web Services, you are actually installing two Web services: the XMLA Web services that were available with SAS 9.1, and the new WebServiceMaker Web service that is used create generated Web services. △

2 Write a SAS program to use as a stored process with Web services. See “Writing SAS Programs for XMLA Web Services” on page 9.

3 Define a stored process server, if one is not already defined.

4 Define a stored process by using SAS Management Console. Use **`XMLA Web Service`** as a keyword and **`Stream`** as the stored process output type when defining a stored process to be called by an XMLA Web service.

Use XMLA Web Services

On the client side, perform the following steps to use XMLA Web services:

1 Locate the Web Service Description Language File (WSDL). You can access the WSDL for a SAS BI Web Service by appending a ’?WSDL’ onto the service endpoint.

2 Write the code for the client application that uses either the Discover method or the Execute method to call the Web service.

3 Run the code.

For XMLA Web services, the SAS code that implements the Web service, the metadata, and the client code that calls the Web service must all be synchronized. The following table shows how to synchronize these items:

Table 1.1 Items to Synchronize

Item	**SAS Program**	**Metadata**	**Client Code**
Name	The name of the file that contains the SAS code.	Associates the name of the SAS Stored Process with the name of the file.	`<StoredProcess name= 'MyStoredProcess'>`
Input Data	Reads XML from the fileref. `libname instream xml;`	The name of the fileref, which must match the name of the data source.	`<Stream name='instream'> <XMLDataFromClientHere... </Stream>`
Input Parameters	Macros. `&tablename`	The parameter name is specified in the metadata. Parameters are treated as strings, regardless of the type that is specified in the metadata.	`<Parameter name='tablename'> myParam</Parameter>`
Output Data	Writes output to the _WEBOUT fileref as XML. `libname _WEBOUT xml xmlmeta= &_XMLSCHEMA;`	Designates the output as 'Streaming'.	Uses the XML that is returned.

Use Generated Web Services

Follow these steps to use generated Web services:

1 Generate a new Web service:

 a Determine URL of the Web Service Maker endpoint.

 b In SAS Management Console, select a set of stored processes and then select **Actions ▶ Deploy As Web Service** to generate a new Web service that can be used to call the selected stored processes.

 In the **`Web Service Maker URL`** field of the Deploy as Web Service wizard, type the endpoint URL or select an existing URL. The user who performs this action should belong to the SAS BI Web Services Users metadata group so that the new Web service can be stored in metadata. The metadata is created only if the service is successfully deployed in the target container. The new Web service will contain one operation for each stored process that you selected.

 Upon successful deployment, a message displays that tells you the endpoint URL for the newly deployed Web service.

2 Create clients to call the Web service.

A Web service can be created with multiple operations in it. Each operation corresponds to a stored process, and has the same name as the stored process, unless there is a naming conflict. If the name of the stored process conflicts with another name, then a new operation name is created.

Overview of Security for Web Services

A default installation of SAS BI Web Services for Java or .NET is not highly secure. The default security mechanism is SAS authentication. All requests and responses are sent as clear-Text. If users want to authenticate as a specific user, then they can send a user name and password as clear-Text as part of the WS-Security headers. Authentication is performed by authenticating client credentials at the SAS Metadata Server. Whenever user names and passwords must be sent as clear-Text, SSL should be enabled to provide transport layer security.

If you are using XMLA Web services or generated Web services, an anonymous user can be configured. Anonymous users cannot use the Web Service Maker; credentials must always be provided to use the Web Service Maker. If you are using XMLA Web services, you can pass user credentials as XMLA properties in the payload.

SAS BI Web Services can be secured by using Web authentication. This provides a way for SAS BI Web Services to identify the calling user by using basic Web authentication. The following two types of Web authentication can be configured:

- WS-Security message-level security
- HTTP transport-level security

Note: Web authentication can be used with both XMLA Web services and generated Web services but cannot be used with the Web Service Maker Web service. The Web Service Maker must be able to authenticate one-time-passwords that are generated by SAS Management Console clients. △

SAS BI Web Services for .NET can be secured by using Web Services Enhancements 3.0 for Microsoft .NET, which enables support for the latest security and interoperability standards for Web services. For detailed information about using Web Services Enhancements 3.0, WS-Security, and WS-Policy to secure SAS BI Web Services, see the *SAS Intelligence Platform: Web Application Administration Guide*. The current release of SAS BI Web Services for Java does not support WS-Policy.

Consult with your administrator to determine how Web services are configured at your site and how you can invoke them. For more information about setting up Web service security, see the *SAS Intelligence Platform: Web Application Administration Guide*.

Understanding Error Codes

Errors generally fall into one of five categories, and are assigned the appropriate error code for that category. The following table describes these error codes:

Table 1.2 Error Codes

Error Code	Description
1000	Specifies an invalid user name or password (the client application might want to re-prompt the user for credentials).
2000	Specifies a client error (the client application might want to pass in different parameters). This error might occur for one of the following reasons: □ invalid prompt value □ required parameter is missing □ invalid request against schema □ invalid stored process name (for XMLA Web services only) □ no ReadMetadata permission for the stored process
3000	Specifies a SAS error. This error is generated when the stored process generates a SYSCC macro variable that is not listed in the AcceptableSyssc configuration option. An additional attribute is added to indicate that the actual error number that SYSCC was set to. The SYSMSG string is also included in the message.
4000	Specifies a configuration error. This indicates a problem that the administrator of the service should be notified about. The administrator should be able to examine logs on the service to determine the cause of this error. This error might occur for one of the following reasons: □ invalid default credentials for the anonymous user □ invalid trusted credentials □ metadata server or stored process server is unreachable □ invalid configuration file
5000	Specifies a timeout error. This error occurs if the user configures SAS BI Web Services with a stored process timeout and the execution of a stored process exceeds this timeout.

Note: Before SAS 9.2, XMLA returned an error code of `99` for almost all errors. △

The following code is an example of a generated SOAP fault that has an error code of **4000**:

```
<SOAP-ENV:Fault>
   <faultcode>Server</faultcode>
   <faultstring>The XML for Analysis provider encountered an error</faultstring>
   <faultactor>XML for Analysis Provider>XML for Analysis Provider</faultactor>
   <detail>
      <sas:Fault code="4000">
         <sas:Exception message="The configured credentials are invalid.">
            <sas:Exception message="The config file contains invalid metadata
credentials."/>
            <sas:Exception message="The user 'anon' is unknown.">
               <sas:Exception message="'anon' is not defined in metadata."/>
            </sas:Exception>
         </sas:Exception>
      </sas:Fault>
   </detail>
</SOAP-ENV:Fault>
```

CHAPTER 2 Writing SAS BI Web Services Using XMLA

Writing SAS Programs for XMLA Web Services

To use the Web service to call your SAS code, you must configure your SAS code as a stored process. A stored process is a SAS program that is stored on a server and can be executed by requesting applications. Any stored process can be deployed as a generated Web service. However, stored processes that are used with XMLA Web services need to conform to rules that enable the Web service to receive data from the client and return data to the client.

You can author a stored process manually by using SAS or a text editor to write the code and then registering the program through SAS Management Console. Alternatively, you can use a program such as SAS Enterprise Guide or another SAS code generator to author a stored process using the point-and-click method. Use the following modifications to make a stored process that can be used with SAS BI Web Services. Keep in mind that XMLA Web services can return data only; no images can be returned.

Note: XMLA Web services in SAS 9.2 will work only with SAS 9.2 stored process metadata and SAS 9.2 Stored Process Servers. △

The following list explains unique details about stored processes that are used with XMLA Web services:

- The data that is returned by the stored process must be XML. Web service stored processes produce streaming results, which means that the SAS program writes output to _WEBOUT, typically by using the following LIBNAME statement:

  ```
  libname _WEBOUT xml xmlmeta=&_XMLSCHEMA;
  ```

- For XMLA Web services, the %STPBEGIN or %STPEND macros are not used in the stored processes. These macros set up Output Delivery System (ODS) statements for the stored process, but Web services do not use ODS.
- The _XMLSCHEMA macro is unique to XMLA Web services. This macro is passed to the SAS program when it is invoked from the Web service. The _XMLSCHEMA macro is set to one of three values depending on the Content property that gets passed to the Execute method. The possible values for _XMLSCHEMA are Schema, SchemaData (which is the default), or Data. For example, the following code causes SAS to write both the XML schema and the data into the libref _WEBOUT:

  ```
  libname _WEBOUT xml xmlmeta=SchemaData;
  ```

  A libref uses a fileref of the same name when a source is not specified in the LIBNAME statement. For example, the following code causes the libref, called _WEBOUT, to read from the fileref called _WEBOUT:

  ```
  libname _WEBOUT xml xmlmeta=_XMLSCHEMA;
  ```

  For Web services, SAS defines the filerefs for _WEBOUT and the input parameter streams before invoking the SAS code.

  Note: Applications should not try to write multiple data sets into a library when a schema is being created. △
- Data sources are unique to Web services. Data sources are defined when you are registering the stored process metadata. There are two types of data sources:
  - generic streams, which are most similar to the input streams that were used before SAS 9.2.
  - XML streams, which can come with a schema or without a schema. If a schema is provided for a generated Web service, then that schema is inserted in the WSDL for the service. If no schema is provided, then **xs:any** is inserted in the schema. Having a schema defined makes it easier for client applications to call a service. The SAS code needs to be written to create XML that is valid according to the schema that is defined in the metadata.

The following example code displays a stored process that is used as a Web service:

```
libname instream xml;
libname _WEBOUT xml xmlmeta=&_XMLSCHEMA;

proc means data=instream.&tablename
   output out=_WEBOUT.mean;
run;
```

The first LIBNAME statement in the sample code defines the data source. This code corresponds to the definition of the data source in the Stored Process Properties dialog box in SAS Management Console. The fileref of the data source is **instream**. In this example, the data source provides the data to run PROC MEANS against.

The second LIBNAME statement in the sample code defines the output for the stored process as streaming output, or _WEBOUT. In the Stored Process Properties dialog box, **Stream** is specified as the type of output on the **Execution** tab of the Stored Process Properties dialog box.

The **&tablename** declaration in the sample code defines a parameter called tablename. In the Stored Process Properties dialog box, this parameter is specified through the New Prompt dialog box, and can be modified using the Edit Prompt dialog box. In this example, tablename is a text parameter that specifies the name of the table to run PROC MEANS against.

Note: The dialog boxes mentioned in the previous example are available from both the Stored Process Properties dialog box and the New Stored Process Wizard, which are both part of SAS Management Console. For more information about using SAS Management Console to define metadata for stored processes, see the product Help. △

Discover Method

Overview of the Discover Method

The Discover method retrieves information, such as stored process metadata or a list of available data sources, from the SAS Metadata Repository. The Discover method returns a list of all the stored processes that have the keyword "XMLA Web Service" on the SAS Metadata Server. The SAS Stored Process Server is not invoked to service the Discover call.

Here is the syntax for the Discover method:

```
Discover (
   [in] RequestType As EnumString,
   [in] Restrictions As Restrictions,
   [in] Properties As Properties,
   [out] Result  As Rowset)
```

RequestType

Overview of RequestType

RequestType is a required parameter for the Discover method. The value of RequestType is an enumeration value that corresponds to a return rowset. The RequestType parameter specifies the type of information to be returned by the Discover request.

There are two main request types that are normally used with SAS BI Web Services: DISCOVER_DATASOURCES and STOREDPROCESS_PARAMETERS. DISCOVER_DATASOURCES and STOREDPROCESS_PARAMETERS both return a list of the stored processes that can be invoked. DISCOVER_DATASOURCES is a standard XMLA request type that returns a list of available data sources for the server or Web service so that you can select a data source with which to connect. The information that is returned by the DISCOVER_DATASOURCES request type includes the following information:

- the name and a description of the data source
- a URL to connect to the data source, the name, and data type of the provider
- the type of security mode that the data source uses, as well as any additional information that is needed to connect to the data source

STOREDPROCESS_PARAMETERS is a request type that is specific to SAS. This request type returns a list of all the available stored processes along with a list of the parameters that are specified in each stored process.

Other request types that might be useful with SAS BI Web Services are DISCOVER_PROPERTIES and DISCOVER_SCHEMA_ROWSETS. DISCOVER_SCHEMA_ROWSETS returns a list of all the available request types along with their enumeration values and other information. For more information about what the DISCOVER_PROPERTIES request type returns, see "Properties" on page 14.

Note: Although the SAS XMLA Stored Process provider supports the DISCOVER_KEYWORDS, DISCOVER_LITERALS, and DISCOVER_ENUMERATORS request types, these request types are not useful for calling stored processes. △

DISCOVER_DATASOURCES

The SAS BI Web Service returns one data source for each stored process that has been defined in the metadata for use with Web services.

For each returned stored process, the returned rowset contains:

DataSourceName
: specifies the name of the stored process, as specified in SAS Management Console. For example,

```
/Samples/Stored Processes/
   Sample: MEANS Procedure Web Service
```

DataSourceDescription
: specifies the description of the stored process, as specified in SAS Management Console. For example,

```
(PROC MEANS Stored Process that can be invoked by
   the SAS BI Web Services for Java/.Net mid-tier.)
```

URL
: specifies the URL to invoke the XMLA methods. This is usually the same as the URL that is used to invoke this Discover method. For example,

```
http://yourserver/xmla/SASSPSProvider/sasxmla.asmx
```

DataSourceInfo
: specifies which data source to use. The SAS Stored Process Server data source is "Provider=SASSPS;".

ProviderName
: specifies the provider behind the data source. For the SAS Stored Process Server, this is the SAS XML for Analysis Stored Process Provider.

ProviderType
: specifies the type of provider that is behind the data source. The Stored Process Service supports only Tabular Data Provider.

AuthenticationMode
: specifies the authentication required for the given data source (that is, indicates whether a user name and password are required). SAS BI Web Services for .NET return "Authenticated" only if a user name and password are required. SAS BI Web Services for Java always return "Authenticated," meaning that you are required to authenticate to the SAS Metadata Repository whether you pass in credentials or use only default credentials that are configured by the administrator.

STOREDPROCESS_PARAMETERS

STOREDPROCESS_PARAMETERS is a custom request type that is used by the SAS Stored Process Service provider only. It returns metadata describing the parameters that are necessary to call the stored process. A stream parameter is always a required parameter and it never has a default. This does not mean that you are required to have a stream parameter for each stored process, but it means that any stream parameters that are defined for the stored process must be provided when the stored process is called using the Execute method.

For each returned stored process, the returned rowset contains:

StoredProcessName
: specifies the name of the stored process.

Parameters
: specifies a container that includes all of the parameters for the stored process.

Parameter
: specifies a container that includes all of the details for a stored process parameter.

Name
: specifies the name of the stored process parameter.

Description
: specifies the description of the stored process parameter.

Type
: specifies the parameter type. The possible parameter types for XMLA Web services are string, multi-line text, Boolean, integer, float, color, time, timestamp, and date. (XMLA Web services do not support advanced prompt types such as data source, data source item, OLAP member, data library, ranges, and prompts with multiple value types.) Note that all parameters are passed to SAS as macro variables, so the SAS program does not know the parameter type that is specified in the metadata. For more information about how to format parameter values, see "Using Prompts with Generated Web Services" on page 26.

Required
: specifies whether the stored process parameter is required.

Default
: specifies a default value for the stored process parameter.

Streams
: specifies a container that includes all of the data sources for the stored process.

Stream
: specifies a container that includes all of the details for a stored process data source.

The following is an example of the response for a stored process that takes a single string and a single stream as input:

```
<row xmlns="urn:schemas-sas-com:xml-analysis:rowset">
   <StoredProcessName>
      /BIP Tree/copyintoout</StoredProcessName>
   <Parameters>
      <Parameter>
         <Name>inputname</Name>
         <Description>A simple string that we are
            passing as a parameter.</Description>
         <Required>true</Required>
         <Default />
```

```
            <Type>String</Type>
         </Parameter>
      </Parameters>
      <Streams>
         <Stream>
            <Name>DataName</Name>
            <Description>This stream does allow
               multi-pass reads, so you do not have to
               use an XMLMap.</Description>
         </Stream>
      </Streams>
   </row>
```

Restrictions

You can use the Restrictions parameter to filter which results get returned from a call to the Discover method. The restriction name specifies a column in a rowset that you restrict. The restriction value specifies which data to restrict in the column. Use the DISCOVER_SCHEMA_ROWSETS request type to get restriction information about the rowsets that are available in each request type. The DISCOVER_SCHEMA_ROWSETS request type returns a list of all the request types that are supported by the provider, along with restriction information and descriptions for each request type.

The Restrictions parameter is required in the Discover method, but it can be empty. Invalid values for restrictions are ignored.

The following RestrictionList element restricts a call to Discover STOREDPROCESS_PARAMETERS based on the name of the stored process:

```
<RestrictionList
   xmlns="urn:schemas-microsoft-com:xml-analysis">
   <StoredProcessName>
      /Samples/Stored Processes/
         Sample: MEANS Procedure Web Service
   </StoredProcessName>
</RestrictionList>
```

Properties

The Properties parameter enables you to specify properties of the Discover method, such as the return format of the result set or the timeout value.

Use the DISCOVER_PROPERTIES request type to get information about properties that are available for each request type and the values that are associated with those properties. The DISCOVER_PROPERTIES request type returns information about both standard and provider-specific properties. The returned information includes the name, description, data type, access, and current value of each property. The information also shows whether each property is required.

The following table contains a list of properties and property information, including sample values, that the DISCOVER_PROPERTIES request type returns. The value of PropertyType for each of these properties is **`string`**.

Table 2.1 Values for the Properties Parameter

PropertyName	**PropertyDescription**	**PropertyAccessType**	**Value**
Content	Specifies the content of the XML result: None, Schema, Data, or Both.	ReadWrite	SchemaData
StateSupport	Specifies the support for state maintenance that is offered by the provider: None or Sessions.	Read	Sessions
UserName	Specifies the user name to use for database authentication.	ReadWrite	
Password	Specifies the password to use for database authentication.	Write	
Domain	Specifies the domain to use for database authentication.	ReadWrite	
ProviderName	Specifies the name of the XML for Analysis provider.	Read	SAS XML for Analysis StoredProcess Provider
ProviderVersion	Specifies the version of the XML for Analysis provider.	Read	1.0
Format	Specifies the format of the XML result: Tabular or Multidimensional.	Read	Tabular
DataSourceInfo	Specifies the identifying information that is required to retrieve data from a data source.	ReadWrite	Provider=SASSPS
Timeout	Specifies the number of seconds until a request fails due to a timeout.	Read	

You can list properties in any order. The Properties parameter is required in the Discover method. The only call to the Discover method that can have empty properties is DISCOVER_DATASOURCES. All other request types require at least DataSourceInfo to be specified, such as:

```
<PropertyList
   xmlns="urn:schemas-microsoft-com:xml-analysis">
   <DataSourceInfo>
      Provider=SASSPS
   </DataSourceInfo>
</PropertyList>
```

To cause a call to Discover to execute under a specific user's identity, a UserName and Password property can be included in the PropertyList element, such as:

```
<PropertyList xmlns="urn:schemas-microsoft-com:xml-analysis">
   <DataSourceInfo>
      Provider=SASSPS
   </DataSourceInfo>
   <UserName>username</UserName>
   <Password>password</Password>
</PropertyList>
```

If you choose to include the UserName or Password properties, it is important to ensure that access to your Web service is secure and encrypted. For more information, see the *SAS Intelligence Platform: Web Application Administration Guide*.

Result

The Result parameter is required. This parameter specifies the result set that the provider returns. The information that is returned can vary according to which values are used in the RequestType, Restrictions, and Properties parameters.

Execute Method

Overview of the Execute Method

Client applications of the Web service call the Execute method to run a SAS Stored Process.

When an application calls the Execute method, the Web service performs the following actions:

- receives the call and validates the SOAP request against the WSDL.
- validates the command against the command schema.
- searches in the SAS Metadata Server to find the SAS server to connect to that can service the request. If the user name and password are provided in the Properties parameter, then they are used to connect to the SAS Metadata Server. The credentials to use when connecting to the SAS Stored Process Server are obtained from the metadata.
- invokes the SAS code that represents the stored process on the SAS Stored Process Server.
- checks the value of the SYSCC macro in SAS. If the SYSCC macro has a non-zero value, then the Web service throws a SOAP fault and includes the value of SYSMSG in the fault.
- returns all data that was written to _WEBOUT.

Here is the syntax for the Execute method:

```
Execute (
   [in] Command As Command,
   [in] Properties As Properties,
   [out] Result As Resultset)
```

Command

The Execute method takes the Command and Properties parameters as input. Both of these parameters are in XML.

The following code shows the command passed to the Execute method:

```
<StoredProcess name="MyStoredProcess">
   <Stream name="DataName"> <myXML>data</myXML> </Stream>
   <Parameter name="InputName">myData</Parameter>
</StoredProcess>
```

When the previous code is passed to the Execute method, the SAS code has a macro defined whose name corresponds to the String parameter:

```
%LET InputName=myData
```

The SAS code also has a libref assigned that corresponds to the name of the Data parameter:

```
libname DataName;
```

The SAS program should write output to the pre-assigned fileref _WEBOUT. Most applications do this by using the XML LIBNAME engine, as follows:

```
libname _WEBOUT xml xmlmeta=&_XMLSCHEMA;
data _WEBOUT.a;
```

Properties

The Properties parameter enables you to specify properties of the Execute method. Properties describe how to invoke the Command parameter. Calling applications specify the SAS Stored Process Service Provider to be used in DataSourceInfo, as shown in the following example:

```
<PropertyList>
   <DataSourceInfo>
      Provider=SASSPS;
   </DataSourceInfo>
</PropertyList>
```

Use the DISCOVER_PROPERTIES request type in the Discover method to get information about properties that are available for each request type and the values that are associated with those properties. The DISCOVER_PROPERTIES request type returns information about both standard and provider-specific properties. The returned information includes the name, description, data type, access, and current value of each property. The information also shows whether each property is required.

You can list properties in any order. The Properties parameter is required in the Discover method, but it can be empty. The Properties parameter must be specified for the Execute method, and must include at least the DataSourceInfo property.

Note: After you have selected a data source from the DISCOVER_DATASOURCES rowset, set the DataSourceInfo property in the Properties parameter, which is sent to the server using the Command parameter by the Execute method. Do not attempt to write your own value for the DataSourceInfo property. Use a value only from the DISCOVER_DATASOURCES rowset. △

To cause the execute method to run under a specific user's identity, a UserName and Password property can be included in the PropertyList element, such as:

```
<PropertyList xmlns="urn:schemas-microsoft-com:xml-analysis">
   <DataSourceInfo>
      Provider=SASSPS
   </DataSourceInfo>
   <UserName>username</UserName>
   <Password>password</Password>
</PropertyList>
```

If you choose to include the UserName or Password properties, it is important to ensure that access to your Web service is secure and encrypted. For more information, see the *SAS Intelligence Platform: Web Application Administration Guide*.

Result

The Result parameter is required. This parameter specifies the result set that the provider returns. The information that is returned can vary according to which values are used in the Command and Properties parameters.

Sample PROC MEANS Using SAS BI Web Services

Sample Overview

This sample shows how to write, define, and invoke a sample stored process that can be used with SAS BI Web Services. This example is for an XMLA Web service. You can access other sample Web services in the samples database at **support.sas.com**.

Write the Stored Process

The following SAS code is a sample stored process called **stpwsmea.sas**. This program is installed with SAS Integration Technologies; by default it is located in **C:\Program Files\SAS\SAS 9.1\inttech\sample**.

```
%put &tablename

libname _WEBOUT xml xmlmeta = &_XMLSCHEMA;
libname instream xml;

proc means data=instream.&tablename
   output out=_WEBOUT.mean;
run;

libname _WEBOUT clear;
libname instream clear;
```

Define the Metadata

The stored process must be defined on a SAS Metadata Server that is used by SAS BI Web Services in order to determine how and where to run the stored process. Stored process metadata is defined by using SAS Management Console. The tables in this section show the values for each field in the New Stored Process Wizard in SAS Management Console.

Note: If you have previously installed the SAS Stored Process sample metadata as part of the SAS Deployment Wizard or the Web Infrastructure Platform installation, then you might not need to re-create the metadata for the "Sample: MEANS Procedure Web Service" sample stored process. The sample metadata should already be available from the **`/Products/SAS Intelligence Platform/Samples`** folder. If you do not have the sample metadata, you can define the metadata for the stored process on your SAS Metadata Server by performing the following steps. △

1 Open SAS Management Console and connect to the appropriate metadata server.

2 From the SAS Management Console navigation tree, select the folder under which you would like to create the new stored process. (If you would like to create a new folder, you can select the location in the navigation tree in which you want to add the new folder, and then select **Actions ▶ New Folder** from the menu bar to open the New Folder Wizard. Follow the wizard instructions to create the new folder.)

3 After you select the folder in which you want to add a new stored process, select **Actions ▶ New Stored Process** from the menu bar. The New Stored Process Wizard displays.

4 On the first page of the New Stored Process Wizard, enter the following values in their corresponding fields for the sample Web service:

Table 2.2 Field Values for the New Stored Process Wizard

Field	Value
Name	Sample: MEANS Procedure Web Service
Keywords	XMLA Web Service

Note: To add the keyword, click **`Add`** to open the Add Keyword dialog box, then enter the name of the keyword. Click **`OK`**. Adding a description and roles for the stored process are optional. △

5 Click **`Next`**.

6 Enter the following values in their corresponding fields for the sample Web service:

Table 2.3 Values for the Sample Web Service

Field	Value
SAS server	SASApp - Logical Stored Process Server
Source code repository	C:\Program Files\SAS\SASFoundation\9.2\inttech\sample
Source code file	stpwsmea.sas
Results	Stream

Click **`Next`**.

7 Click **New Prompt** to add an input parameter to the stored process.

8 On the **General** tab, enter the following values in their corresponding fields for the sample Web service:

Table 2.4 Values for the Prompt

Field	Value
Name	tablename
Displayed text	tablename

9 Select the **Requires a non-blank value** check box. Entering a description is optional.

10 On the **Prompt Type and Values** tab, enter the following values in their corresponding fields for the sample Web service:

Table 2.5 Values for the Prompt

Field	Value
Prompt type	Text
Method for populating prompt	User-entered value
Number of values	Single value
Text type	Single line
Default value	InData

11 Click **Next**.

12 Click **New** to open the New Data Source dialog box, where you must define the data source.

a Enter the following values in their corresponding fields for the sample Web service:

Table 2.6 Values for the New Data Source

Field	Value
Type	XML Stream
Label	instream
Fileref	instream
Expected content type	text/xml

b You must also select the **Allow rewinding stream** check box in the New Data Source dialog box. Otherwise, an XMLMap would need to be specified on the XML LIBNAME statement to define the XML schema for **instream**.

c Click **OK** to save the data source definition.

13 Review your stored process information, and click **Finish** to define the metadata for the stored process.

Invoke the Stored Process

SOAP Request

The stored process that we just created can be invoked by SAS BI Web Services for Java and .NET middle-tier clients. A Web service client invokes the middle-tier Web service with an **`Execute()`** command. The SOAP request body, or client code, follows:

```
<soap-env:Body>
   <Execute>
      <Command>
         <StoredProcess
            name="/Samples/Stored Processes/Sample:
            MEANS Procedure Web Service">
            <Parameter name="tablename">InData</Parameter>
            <Stream name="instream">
               <Table>
                  <InData>
                     <Column1>1</Column1>
                     <Column2>20</Column2>
                     <Column3>99</Column3>
                  </InData>
                  <InData>
                     <Column1>50</Column1>
                     <Column2>200</Column2>
                     <Column3>9999</Column3>
                  </InData>
                  <InData>
                     <Column1>100</Column1>
                     <Column2>2000</Column2>
                     <Column3>1000000</Column3>
                  </InData>
               </Table>
            </Stream>
         </StoredProcess>
      </Command>
      <Properties>
         <PropertyList>
            <DataSourceInfo>Provider=SASSPS;</DataSourceInfo>
         </PropertyList>
      </Properties>
   </Execute>
</soap-env:Body>
```

SOAP Response

After you run the client code, the resulting SOAP response body is as follows:

```
<soap-env:Body>
   <ExecuteResponse>
      <return>
         <root>
            <TABLE>
```

```
               <MEAN>
                  <_TYPE_> 0 </_TYPE_>
                  <_FREQ_> 3 </_FREQ_>
                  <_STAT_> N </_STAT_>
                  <COLUMN3> 3 </COLUMN3>
                  <COLUMN2> 3 </COLUMN2>
                  <COLUMN1> 3 </COLUMN1>
               </MEAN>
               <MEAN>
                  <_TYPE_> 0 </_TYPE_>
                  <_FREQ_> 3 </_FREQ_>
                  <_STAT_> MIN </_STAT_>
                  <COLUMN3> 99 </COLUMN3>
                  <COLUMN2> 20 </COLUMN2>
                  <COLUMN1> 1 </COLUMN1>
               </MEAN>
               <MEAN>
                  <_TYPE_> 0 </_TYPE_>
                  <_FREQ_> 3 </_FREQ_>
                  <_STAT_> MAX </_STAT_>
                  <COLUMN3> 1000000 </COLUMN3>
                  <COLUMN2> 2000 </COLUMN2>
                  <COLUMN1> 100 </COLUMN1>
               </MEAN>
               <MEAN>
                  <_TYPE_> 0 </_TYPE_>
                  <_FREQ_> 3 </_FREQ_>
                  <_STAT_> MEAN </_STAT_>
                  <COLUMN3> 336699.333 </COLUMN3>
                  <COLUMN2> 740 </COLUMN2>
                  <COLUMN1> 50.3333333 </COLUMN1>
               </MEAN>
               <MEAN>
                  <_TYPE_> 0 </_TYPE_>
                  <_FREQ_> 3 </_FREQ_>
                  <_STAT_> STD </_STAT_>
                  <COLUMN3> 574456.555 </COLUMN3>
                  <COLUMN2> 1094.89726 </COLUMN2>
                  <COLUMN1> 49.5008417 </COLUMN1>
               </MEAN>
            </TABLE>
         </root>
      </return>
   </ExecuteResponse>
</soap-env>
```

CHAPTER 3
Using Generated Web Services

What Are Generated Web Services?

Starting with SAS 9.2, you can select a set of stored processes in SAS Management Console and deploy them to the Web Service Maker. The Web Service Maker generates a new Web service that contains one operation for each stored process that you selected. A new Web Service Description Language File (WSDL) document is generated when you deploy the Web service. The generated WSDL document contains all the information that would be obtained if you were using the Discover method with an XMLA Web service.

Having all the type information in the WSDL is better suited to most client applications, and also makes things simpler for the developer. Making the WSDL more specific to the actual parameters instead of having a generic interface enables you to simplify the request XML. Making the WSDL more specific also makes the Web service more easily consumed by standard Web service client applications such as BizTalk, InfoPath, Word, SharePoint, Excel, AJAX, and WebSphere.

Generated Web services also support attachments. Attachments enable you to send non-XML (binary) data with a SOAP request or response. Most stored processes generate binary output (because they create reports that contain HTML and GIF images). MTOM is currently supported by .NET and the Apache Axis2 Web Service stack.

Note: If you are using SAS BI Web Services for .NET, attachments have a content-type of **`application/octet-stream`**. The actual content-type of the attachment is specified in the SOAP body. △

For information about the basic steps for using generated Web services, see "Use Generated Web Services" on page 4.

Differences Between XMLA Web Services and Generated Web Services

The major differences between XMLA Web services and generated Web services are:

- *Consumption capabilities.* Generated Web services have a WSDL that is customized for each stored process that is in the service. This enables client application developers to create proxies that can create and read the XML documents that are exchanged with the service. XMLA services are described in the Discover call, so proxies must be manually created by the developer for calling the service.
- *Attachments.* XMLA Web services can process XML only. Generated Web services can read and write binary information by using attachments. For example, this means you can return graphs that are generated by ODS by using generated services.
- *Output parameters.* The only allowed output from XMLA is the _WEBOUT stream. Generated services can return output parameters, the _WEBOUT stream, packages (by using attachments), and data targets.
- *Deployment.* To deploy a stored process for XMLA, you set the **`XMLA Web Service`** keyword. To deploy a stored process for a generated service, you use a wizard in the folder view of SAS Management Console.
- *Metadata.* Generated Web services have metadata that can be accessed using SAS Management Console.

Using Attachments with Generated Web Services

Streaming attachments can be defined in metadata as data sources (input attachments) and data targets (output attachments). Two types of streaming attachments are available:

XML stream
: specifies an attachment that is in-lined in the payload of the SOAP request or SOAP response. You can also specify a schema for this data. The schema is included in the generated WSDL.

Note: You can specify single streaming output the same way you do with the XMLA Web service by selecting **`Stream`** as the result capability. However, using data targets provides more flexibility because you can provide the name of the attachment as well as provide a schema that matches your expected data. △

generic stream
: specifies an out-of-band binary attachment that is included with the SOAP request or SOAP response in one of the following ways:

  - If the attachment data is small, it can be included directly in the payload and encoded as Base64 binary data.
  - If the attachment data is not small, then it is included out-of-band from the payload as a MIME multi-part related attachment where it is referenced from the payload via MTOM XOP/Include or SOAP with Attachments references (swaRef).

The following code is an example of a schema definition for a generated Web service that expects one generic (binary) stream as an output response:

```
<element name="stpAllParm1Response">
   <complexType>
```

```
        <sequence>
           <element name="stpAllParm1Result">
              <complexType>
                 <sequence>
                    <element maxOccurs="1" minOccurs="0" name="Streams">
                       <complexType>
                          <sequence>
                             <element maxOccurs="1" minOccurs="0" name="myAttachment">
                                <complexType>
                                   <sequence>
                                      <element name="Value" type="base64Binary"/>
                                   </sequence>
                                   <attribute name="contentType" type="string"/>
                                </complexType>
                             </element>
                          </sequence>
                       </complexType>
                    </element>
                 </sequence>
               </complexType>
           </element>
        </sequence>
     </complexType>
  </element>
```

In this generated schema, **`myAttachment`** is the name of the element that represents the output attachment. This name is defined by the user in metadata. This element is a container for the actual value of that attachment. The content type of the attachment can be returned as an attribute to further clarify the content of the data within the attachment.

Package type output is also supported. This type of output produces one or more attachments and packages them together as a single entity. To enable this type of output, select **`Package`** as the result capability for the stored process.

Attachment definitions in metadata provide a means to establish a contract between all parties involved in a Web service request. The Web Service Maker Web service generates a WSDL and schema based on metadata definitions that provide a contract between the client and generated Web service. The generated Web service enforces that all required attachments are sent in the request. The SAS code that executes on the SAS server must be written in accordance to the metadata definitions that it is representing; otherwise, problems might occur (for example, not reading the correct stream, and so on) resulting in SAS errors. If a SAS error occurs, the generated Web service returns a SOAP fault to the client.

Using Prompts with Generated Web Services

When defining stored process parameters, the prompt type definitions are mapped to Web service schema as follows:

Table 3.1 How Prompt Types Map to Web Service Schema

Prompt Type	**Parameter Type in Generated WSDL**
Text, Date, Time, Color, Data source, File or directory, Data library	**xs:string**
Numeric	**xs:int** or **xs:double**
Ranges have **lowerBound** and **upperBound** elements	**xs:***type* (where *type* is the appropriate value, such as **int** or **dateTime**, from this table) **xs:string** (for **lowerBound** and **upperBound** elements)
Timestamp	**xs:dateTime**
Data source item has **path** and **itemName** elements	**xs:string** (for **path** and **itemName** elements)
OLAP member has **label** and **uniqueName** elements	**xs:string** (for **label** and **uniqueName** elements)

The **xs:** prefix in these values is an abbreviation for the namespace being used. This particular abbreviation stands for the standard XML schema namespace, **http://www.w3.org/2001/XMLSchema**. For more information about the XML schema, see **http://www.w3.org/2001/XMLSchema**.

For generated Web services, the WSDL that is generated for Java is different from the WSDL that is generated for .NET. SAS BI Web Services for Java uses facets and restrictions that are based on prompt constraints. These constraints do not appear in the WSDL that is generated for .NET. In both cases, values are validated against the constraints that are defined in metadata. However, SAS BI Web Services for .NET does not support dynamic prompt validation. (Dynamic prompts allow you to look up possible prompt values from a data source such as a SAS data set or information map at run time.)

The following table explains how to format values for the various prompt types:

Table 3.2 Guidelines for Entering Prompt Values (U.S. English Locale)

Prompt Type	**Guidelines**	**Examples**
Text	Enter any character value. Blank spaces and nonprintable characters can be used, but the value cannot consist completely of these characters. Trailing blanks are stored as part of the value and are included when the value is validated against the minimum and maximum length requirements.	□ **`you are here`** □ **`eighty-five`** □ **`Bob`**
Numeric	Enter a standard numeric value. □ If you are working with an integer prompt, then do not use values with decimal places. If you use a value with zeros after the decimal point (for example, **`1.00`**) for an integer prompt, then the zeros and the decimal point will be removed before the value is stored (for example, **`1.00`** will be stored as **`1`**). □ For prompts that allow floating-point values, the unformatted prompt value can contain up to 15 significant digits. Values with more than 15 significant digits of precision are truncated. Note that formatted values can have more than 15 significant digits.	□ **`1.25`** □ **`6000`** □ **`2222.444`**

Prompt Type	Guidelines	Examples
Date	For dates of type Day, enter values in one of the following formats: □ *ddmmmyyyy* □ *ddmonth-nameyyyy* (Java only) □ *mm/dd/yy<yy>* □ *mm.dd.yy<yy>* □ *mm-dd-yy<yy>* □ *mmm/dd/yy<yy>* □ *mmm.dd.yy<yy>* □ *mmm-dd-yy<yy>* □ *mmm dd, yyyy* (Java only) □ *month-name/dd/yy<yy>* (Java only) □ *month-name.dd.yy<yy>* (Java only) □ *month-name-dd-yy<yy>* (Java only) □ *month-name dd, yyyy* □ *day-of-week, mmm dd, yy* (Java only) □ *day-of-week, mmm dd, yyyy* (Java only) □ *day-of-week, month-name dd, yy* (Java only) □ *day-of-week, month-name dd, yyyy* □ *yyyy/mm/dd* (Java only) □ *yyyy.mm.dd* (Java only) □ *yyyy-mm-dd* (Java only) □ *yyyy.mmm.dd* (Java only) □ *yyyy-mmm-dd* (Java only) □ *yyyy.month-name.dd* (Java only) □ *yyyy-month-name-dd* (Java only)	□ **4APR1860** □ **14January1918** □ **12/14/45** □ **02.15.1956** □ **1--1--60** □ **Oct/02/08** □ **JUL.20.13** □ **MAY-13--1924** □ **Oct 05, 2006** □ **February/10/00** □ **March.1.2004** □ **DECEMBER-25--08** □ **SEPTEMBER 20, 2010** □ **FRI, Jan 3, 20** □ **Tuesday, Jan 15, 2008** □ **Monday, January 16, 40** □ **FRIDAY, JANUARY 04, 2008** □ **2041/5/13** □ **2050.07.25** □ **2100--1--1** □ **2009.NOV.02** □ **2400--Aug-8** □ **2101.December.31** □ **1919--APRIL-20**

Prompt Type	Guidelines	Examples
	Here is an explanation of the syntax: *day-of-week* specifies either the first three letters of the day of the week or the full name of the day of the week (the full name of the day must be used for values in .NET). This value is not case sensitive. (That is, the lowercase and uppercase versions of the same character are considered to be the same.) *dd* specifies a one-digit or two-digit integer that represents the day of the month. *mm* specifies a one-digit or two-digit integer that represents the month of the year. *mmm* specifies the first three letters of the full name of the month. This value is not case sensitive. *month-name* specifies the full name of the month. This value is not case sensitive. (That is, the lowercase and uppercase versions of the same character are considered to be the same.) *yy* or *yyyy* specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for a four-digit year range from 1600 to 2400.	

Prompt Type	Guidelines	Examples
	For dates of type Week, enter values in one of the following formats: □ **W***ww yy* □ **W***ww yyyy* (Java only) □ **Week***ww yyyy* Here is an explanation of the syntax: *ww* specifies a one-digit or two-digit integer that represents the week of the year. Valid values range from 1 to 52. *yy* or *yyyy* specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for a four-digit year range from 1600 to 2400.	□ **`W1 08`** □ **`W52 1910`** □ **`Week 20 2020`** □ **`Week 5 2048`**

Prompt Type	Guidelines	Examples
	For dates of type Month, enter values in one of the following formats: □ *mm*/*yy<yy>* □ *mm.yy<yy>* □ *mm-yy<yy>* □ *mmm yy<yy>* (Java only) □ *mmm*/*yy<yy>* □ *mmm.yy<yy>* □ *mmm-yy<yy>* □ *month-name yy* (Java only) □ *month-name yyyy* □ *month-name*/*yy<yy>* (Java only) □ *month-name.yy<yy>* (Java only) □ *month-name-yy<yy>* (Java only) Here is an explanation of the syntax: *mm* specifies a one-digit or two-digit integer that represents the month of the year. *mmm* specifies the first three letters of the full name of the month. This value is not case sensitive. *month-name* specifies the full name of the month. This value is not case sensitive. (That is, the lowercase and uppercase versions of the same character are considered to be the same.) *yy* or *yyyy* specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for a four-digit year range from 1600 to 2400.	□ `12/1828` □ `06.65` □ `7--76` □ `Jul 08` □ `JUN/2010` □ `SEP.20` □ `Oct-2050` □ `August 20` □ `OCTOBER 1975` □ `MARCH/1970` □ `May.13` □ `November-18`

Prompt Type	Guidelines	Examples
	For dates of type Quarter, enter values in the following format: □ *quarter-name* **`quarter`** *yy<yy>* Here is an explanation of the syntax: *quarter-name* specifies the quarter of the year. Valid values are **`1st`**, **`2nd`**, **`3rd`**, and **`4th`**. *yy* or *yyyy* specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for a four-digit year range from 1600 to 2400.	□ **`1st quarter 1900`** □ **`2nd quarter 50`** □ **`3rd quarter 12`** □ **`4th quarter 2060`**

Prompt Type	Guidelines	Examples
	For dates of type Year, enter values in one of the following formats: □ *yy* (Java only) □ *yyyy* Here is an explanation of the syntax: *yy* or *yyyy* specifies a two-digit or four-digit integer that represents the year. To refer to a year that is more than 80 years in the past or 20 years in the future, use four digits. Valid values for a four-digit year range from 1600 to 2400.	□ **1895** □ **86** □ **08** □ **2035**

Prompt Type	Guidelines	Examples
Time	Enter time values in the following format: □ *hh:mm*<*:ss*> <**AM** | **PM**> Here is an explanation of the syntax: *hh* specifies a one-digit or two-digit integer that represents the hour of the day. Valid values range from 0 to 24. *mm* specifies a one-digit or two-digit integer that represents the minute of the hour. Valid values range from 0 to 59. *ss* (optional) specifies a one-digit or two-digit integer that represents the second of the minute. Valid values range from 0 to 59. If this value is not specified, then the value defaults to 00 seconds. **AM** or **PM** (optional) specifies either the time period 00:01 to 12:00 noon (AM) or the time period 12:01 to 12:00 midnight (PM). If this value is not specified and you are using the 12-hour system for specifying time, then the value defaults to **AM**. *Note:* Do not specify **AM** or **PM** if you are using the 24-hour system for specifying time. △	□ **1:1** □ **1:01 AM** □ **13:1:1** □ **01:01:01 PM** □ **22:05**

Prompt Type	**Guidelines**	**Examples**
Timestamp	Enter timestamp values in the following format: □ *yyyy-mm-dd*T*hh:mm:ss* Here is an explanation of the syntax: *yyyy* specifies a four-digit integer that represents the year. Valid values for a four-digit year range from 1600 to 2400. *mm* specifies a one-digit or two-digit integer that represents the month of the year. *dd* specifies a one-digit or two-digit integer that represents the day of the month. *hh* specifies a one-digit or two-digit integer that represents the hour of the day. Valid values range from 0 to 24. *mm* specifies a one-digit or two-digit integer that represents the minute of the hour. Valid values range from 0 to 59. *ss* specifies a one-digit or two-digit integer that represents the second of the minute. Valid values range from 0 to 59.	□ `2012-11-23T15:30:32` □ `2008-09-09T01:01:01`

Prompt Type	Guidelines	Examples
Color	Enter color values in one of the following formats: □ **CX***rrggbb* □ **0x***rrggbb* □ #*rrggbb* Here is an explanation of the syntax: *rr* specifies the red component. *gg* specifies the green component. *bb* specifies the blue component. Each component should be specified as a hexadecimal value that ranges from 00 to FF, where lower values are darker and higher values are brighter.	Bright red □ **CXFF0000** □ **0xFF0000** □ **#FF0000** Black □ **CX000000** □ **0x000000** □ **#000000** White □ **CXFFFFFF** □ **0xFFFFFF** □ **#FFFFFF**
Data source	Enter the name and location of a data source in the following format: □ */folder-name-1/<.../ folder-name-n/ >data-source-name(type)* Here is an explanation of the syntax: */folder-name-1/<.../ folder-name-n/>* specifies the location of the data source. *data-source-name* specifies the name of the data source. *type* is the type of data source. The following values are valid unless otherwise noted: **Table**, **InformationMap**, and **Cube**. Use **InformationMap** for specifying either relational or OLAP information maps.	□ **/Shared Data/Tables/ OrionStar/ Customers(Table)** □ **/Users/MarcelDupree/ My Folder/My Information Map(InformationMap)** □ **/MyCustomRepository/ More Data/ Order_Facts(Table)**

Prompt Type	Guidelines	Examples
File or directory	Enter the name and location of a file or directory in the following format: □ *directory-specification<filename>* Here is an explanation of the syntax: *directory-specification* specifies the location of the file or directory in the file system of a SAS server. *filename* specifies the name of the file. This value is required only if the prompt is a file prompt. Depending on the operating environment that the SAS server runs in, you might need to put a forward slash (/) or a backslash (\\) between the directory specification and the name of the file.	□ **`C:\Documents and Settings\All Users\Documents\myfile.txt`** □ **`\\myserver.internal.com\Documents and Settings\All Users\Documents\myfile.txt`** □ **`\\node1234\Documents and Settings\All Users\Documents`**
Data library	Enter the name and location of a data library in the following format: □ */folder-name-1/<.../folder-name-n/>library-name* **`(Library)`** Here is an explanation of the syntax: */folder-name-1/<.../folder-name-n/>* specifies the location of the library. *library-name* specifies the name of the library.	□ **`/Data/Libraries/Customer Data Library(Library)`** □ **`/MyCustomRepository/More Data/OracleData(Library)`**

Prompt Type	Guidelines	Examples
Data source item	For the path element, enter the path for a data source item in the following format: □ */folder-name-1/<.../ folder-name-n/ >data-source-name(type)* Here is an explanation of the syntax: */folder-name-1/<.../ folder-name-n/>* specifies the location of the data source. *data-source-name* specifies the name of the data source. *type* is the type of data source. The following values are valid unless otherwise noted: **`Table`** or **`InformationMap`**. Use **`InformationMap`** for specifying either relational or OLAP information maps. For the itemName element, enter the name for the data source item in the following format: □ *item-name* Here is an explanation of the syntax: *item-name* specifies the name of the data source item. This is the name of a column in a table or a data item in an information map.	path □ **`/Shared Data/Tables/ MYDATA(Table)`** itemName □ **`Year`**
OLAP member	For the uniqueName element, enter the name of the OLAP member. For the label element, enter the label for the OLAP member.	uniqueName □ **`PRICEAVG`** label □ **`Average Price`**

Note: An anonymous user cannot launch workspace servers. Dynamic prompt validation requires use of workspace servers. Thus, anonymous users will not be able to use all stored processes. △

Prompt definitions in metadata provide a means to establish a contract between all parties that are involved in a Web service request. The Web Service Maker Web service generates a WSDL and schema based on metadata definitions that provide a contract between the client and generated Web service. Mature client programming tools help assist clients in formulating valid requests. The generated Web service also validates client requests via the SAS prompting framework. The SAS code that executes on the

SAS server must be written in accordance to the metadata definitions that it represents. Otherwise, problems occur (for example, reading the wrong input parameter, expecting a different type, and so on) that result in SAS errors. If a SAS error occurs, then the generated Web service returns a SOAP fault to the client.

Sample WSDLs

Sample Parameters

The following table contains names, prompt types, and restrictions for sample parameters for a stored process.

Table 3.3 Sample Stored Process Parameters

Prompt Name	Prompt Type	Restrictions
top_level	Text	Single value
fixed	Text	Read-only values Single value Default value: **`fixed default`**
simple_string	Text	Single value
invisible	Text	Hide from user Single value Default value: **`hidden val`**
default	Text	Single value Default value: **`def val`**
static_list	Text	Multiple ordered values
max_length	Text	Single value Maximum length: **`6`**
mult_entry	Text	Multiple values Maximum value count: **`5`**
text_range	Text range	Default range from: **`aaa`** Default range to: **`zzz`**
req_string	Text	Requires a non-blank value Single value
simple_int	Numeric (integer)	Single value Allows only integer values
fixed_int	Numeric (integer)	Read-only values Single value Allows only integer values Default value: **`12345`**

Prompt Name	**Prompt Type**	**Restrictions**
def_int	Numeric (integer)	Single value Allows only integer values Default value: `12345`
int_list	Numeric (integer)	Multiple ordered values Allows only integer values
int_mult	Numeric (integer)	Requires a non-blank value Multiple values Allows only integer values Minimum value count: `1` Maximum value count: `5`
lim_int	Numeric (integer)	Single value Allows only integer values Minimum value allowed: `1` Maximum value allowed: `99`
req_int	Numeric (integer)	Requires a non-blank value Single value Allows only integer values Default value: `9999`
simple_float	Numeric (double)	Single value
def_float	Numeric (double)	Single value Minimum number of decimal places displayed: `1` Maximum number of decimal places displayed:`3` Minimum value allowed: `1.0` Maximum value allowed: `100.0` Default value: `99.99`
float_list	Numeric (double)	Multiple values Minimum number of decimal places displayed: `1` Maximum number of decimal places displayed: `3`
float_mult	Numeric (double)	Multiple values Maximum number of decimal places displayed: `4` Maximum value count: `5` Maximum value allowed: `999999.0`
lim_float	Numeric (double)	Single value Minimum value allowed: `10.0` Maximum value allowed: `20.0`

Prompt Name	Prompt Type	Restrictions
req_float	Numeric (double)	Requires a non-blank value Single value Default value: **`99.0`**
simple_color	Color	
def_color	Color	Default value: **`CXFF0000`**
fixed_color	Color	Read-only values Default value: **`CX0000FF`**
req_color	Color	Requires a non-blank value Default value: **`CXFFFF00`**
simple_date	Date	Single value
def_date	Date	Single value Default value: **`Today`**
date_list	Date	Multiple values Minimum value allowed: **`October 01, 2007`** Maximum value allowed: **`N days from now (200)`**
date_range	Date range	Minimum value allowed: **`October 01, 2007`** Maximum value allowed: **`N days from now (300)`**
req_date	Date	Requires a non-blank value Single value Include special values: Missing values Default value: **`Week 50 2007`**
simple_time	Time	
fixed_time	Time	Read-only values Default value: **`Current hour`**
def_time	Time	Minimum value allowed: **`N hours ago (1)`** Maximum value allowed: **`N hours from now (1)`** Default value: **`Current hour`**
timerange	Time range	Default range type: Custom Default range from: **`N hours ago (10)`** Default range to: **`N hours from now (1)`**
file1	File or directory	
data_source	Data source	Default value: **`/Stored Processes/CARS(Table)`**

Prompt Name	Prompt Type	Restrictions
data_source_item	Data source item	Single value Default value: **`Make [Make] [/Stored Processes/CARS]`**
data_library	Data library	Default value library: **`/Stored Processes/ WsmSASHelp(Library)`** Default value libref: **`myref`**
olap_member	OLAP member	Single value

Generated WSDL for .NET

If a Web service is generated for a stored process with these sample parameters, the following WSDL is generated for .NET:

```
<wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/AllPromptTypes">
    <s:import namespace="http://support.sas.com/xml/namespace/biwebservices/attachments-9.2" />
    <s:element name="stpAllParm1">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="top_level" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="simple_string" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" default="def val" name="default" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="static_list" type="tns:ArrayOfString" />
          <s:element minOccurs="0" maxOccurs="1" name="max_length" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="mult_entry" type="tns:ArrayOfString" />
          <s:element minOccurs="0" maxOccurs="1" name="text_range" type="tns:GenericRangeSerializerOfString" />
          <s:element minOccurs="0" maxOccurs="1" name="req_string" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="simple_int" type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" default="12345" name="def_int" type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" name="int_list" type="tns:ArrayOfInt" />
          <s:element minOccurs="0" maxOccurs="1" name="int_mult" type="tns:ArrayOfInt" />
          <s:element minOccurs="0" maxOccurs="1" name="lim_int" type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" default="9999" name="req_int" type="s:int" />
          <s:element minOccurs="0" maxOccurs="1" name="simple_float" type="s:double" />
          <s:element minOccurs="0" maxOccurs="1" default="99.99" name="def_float" type="s:double" />
          <s:element minOccurs="0" maxOccurs="1" name="float_list" type="tns:ArrayOfDouble" />
          <s:element minOccurs="0" maxOccurs="1" name="float_mult" type="tns:ArrayOfDouble" />
          <s:element minOccurs="0" maxOccurs="1" name="lim_float" type="s:double" />
          <s:element minOccurs="0" maxOccurs="1" default="99" name="req_float" type="s:double" />
          <s:element minOccurs="0" maxOccurs="1" name="simple_color" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="def_color" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="req_color" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="simple_date" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" default="D0D" name="def_date" type="s:string" />
          <s:element minOccurs="0" maxOccurs="1" name="date_list" type="tns:ArrayOfString" />
          <s:element minOccurs="0" maxOccurs="1" name="date_range" type="tns:GenericRangeSerializerOfString" />
          <s:element minOccurs="0" maxOccurs="1" name="req_date" nillable="true">
            <s:complexType>
              <s:simpleContent>
                <s:extension base="s:string">
              <s:attribute name="missing" use="optional">
                <s:simpleType>
                  <s:restriction base="s:string">
                    <s:minLength value="1" />
                    <s:maxLength value="1" />
                    <s:pattern value="[.A-Z_]" />
                  </s:restriction>
                </s:simpleType>
              </s:attribute>
            </s:extension>
          </s:simpleContent>
        </s:complexType>
      </s:element>
      <s:element minOccurs="0" maxOccurs="1" name="simple_time" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" default="H0H" name="def_time" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="timerange" type="tns:GenericRangeSerializerOfString" />
      <s:element minOccurs="0" maxOccurs="1" name="file1" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="data_source" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="data_source_item">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="path" type="s:string" />
            <s:element minOccurs="1" maxOccurs="1" name="itemName" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element minOccurs="0" maxOccurs="1" name="data_library" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="olap_member">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="uniqueName" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="label" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="ArrayOfString">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="string" nillable="true" type="s:string" />
  </s:sequence>
</s:complexType>
<s:complexType name="GenericRangeSerializerOfString">
  <s:complexContent mixed="false">
    <s:extension base="tns:ARangeTypeSerializer">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="lowerBound" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="upperBound" type="s:string" />
      </s:sequence>
    </s:extension>
  </s:complexContent>
</s:complexType>
<s:complexType name="ARangeTypeSerializer" />
```

```
    <s:complexType name="ArrayOfInt">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="int" type="s:int" />
      </s:sequence>
    </s:complexType>
    <s:complexType name="ArrayOfDouble">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="double" type="s:double" />
      </s:sequence>
    </s:complexType>
    <s:element name="stpAllParm1Response">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" ref="s1:stpAllParm1Result" />
        </s:sequence>
      </s:complexType>
    </s:element>
  </s:schema>
  <s:schema elementFormDefault="qualified" targetNamespace=
      "http://support.sas.com/xml/namespace/biwebservices/attachments-9.2">
    <s:element name="stpAllParm1Result" type="s1:StreamType" />
    <s:complexType name="StreamType">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Value" type="s:base64Binary" />
      </s:sequence>
      <s:attribute name="name" type="s:string" />
      <s:attribute name="contentType" type="s:string" />
    </s:complexType>
  </s:schema>
</wsdl:types>
```

Generated WSDL for Java

If a Web service is generated for a stored process with these sample parameters, the following WSDL is generated for Java:

```
<types>
  <schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/AllPromptTypes"
     xmlns="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://tempuri.org/AllPromptTypes">
       <annotation>
          <documentation>SAS BI Web Services generated schema</documentation>
       </annotation>
       <element name="stpAllParm1">
          <complexType>
             <sequence>
                <element name="parameters" type="tns:stpAllParm1Parameters"/>
             </sequence>
          </complexType>
       </element>
       <complexType name="stpAllParm1Parameters">
          <sequence>
             <element maxOccurs="1" minOccurs="0" name="top_level" type="string"/>
             <element maxOccurs="1" minOccurs="0" name="simple_string" type="string"/>
             <element default="def val" maxOccurs="1" minOccurs="0" name="default" type="string"/>
```

```
<element maxOccurs="1" minOccurs="0" name="static_list">
   <complexType>
      <sequence>
         <element maxOccurs="unbounded" minOccurs="0" name="Item" type="string"/>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="0" name="max_length">
   <simpleType>
      <restriction base="string">
         <maxLength value="6"/>
      </restriction>
   </simpleType>
</element>
<element maxOccurs="1" minOccurs="0" name="mult_entry">
   <complexType>
      <sequence>
         <element maxOccurs="5" minOccurs="0" name="Item" type="string"/>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="0" name="text_range">
   <complexType>
      <sequence>
         <element name="LowerBound" type="string"/>
         <element name="UpperBound" type="string"/>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="1" name="req_string" type="string"/>
<element maxOccurs="1" minOccurs="0" name="simple_int" type="int"/>
<element default="12345" maxOccurs="1" minOccurs="0" name="def_int" type="int"/>
<element maxOccurs="1" minOccurs="0" name="int_list">
   <complexType>
      <sequence>
         <element maxOccurs="unbounded" minOccurs="0" name="Item" type="int"/>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="1" name="int_mult">
   <complexType>
      <sequence>
         <element maxOccurs="5" minOccurs="1" name="Item" type="int"/>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="0" name="lim_int">
   <simpleType>
      <restriction base="int">
         <minInclusive value="1"/>
         <maxInclusive value="99"/>
      </restriction>
   </simpleType>
</element>
```

```
<element default="9999" maxOccurs="1" minOccurs="1" name="req_int" type="int"/>
<element maxOccurs="1" minOccurs="0" name="simple_float" type="double"/>
<element default="99.99" maxOccurs="1" minOccurs="0" name="def_float">
   <simpleType>
      <restriction base="double">
         <minInclusive value="1.0"/>
         <maxInclusive value="100.0"/>
      </restriction>
   </simpleType>
</element>
<element maxOccurs="1" minOccurs="0" name="float_list">
   <complexType>
      <sequence>
         <element maxOccurs="unbounded" minOccurs="0" name="Item" type="double"/>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="0" name="float_mult">
   <complexType>
      <sequence>
         <element maxOccurs="5" minOccurs="0" name="Item">
            <simpleType>
               <restriction base="double">
                  <maxInclusive value="999999.0"/>
               </restriction>
            </simpleType>
         </element>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="0" name="lim_float">
   <simpleType>
      <restriction base="double">
         <minInclusive value="10.0"/>
         <maxInclusive value="20.0"/>
      </restriction>
   </simpleType>
</element>
<element default="99.0" maxOccurs="1" minOccurs="1" name="req_float" type="double"/>
<element maxOccurs="1" minOccurs="0" name="simple_color" type="string"/>
<element default="cxff0000" maxOccurs="1" minOccurs="0" name="def_color" type="string"/>
<element default="cxffff00" maxOccurs="1" minOccurs="1" name="req_color" type="string"/>
<element maxOccurs="1" minOccurs="0" name="simple_date" type="string"/>
<element default="D0D" maxOccurs="1" minOccurs="0" name="def_date" type="string"/>
<element maxOccurs="1" minOccurs="0" name="date_list">
   <complexType>
      <sequence>
         <element maxOccurs="unbounded" minOccurs="0" name="Item">
            <simpleType>
               <restriction base="string">
                  <enumeration value="October 05, 2007"/>
                  <enumeration value="October 31, 2007"/>
               </restriction>
            </simpleType>
         </element>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="0" name="date_range">
   <complexType>
      <sequence>
         <element name="LowerBound" type="string"/>
         <element name="UpperBound" type="string"/>
      </sequence>
   </complexType>
</element>
<element default="Week 50 2007" maxOccurs="1" minOccurs="1" name="req_date" nillable="true">
   <complexType>
      <simpleContent>
         <extension base="string">
            <attribute name="missing">
               <simpleType>
                  <restriction base="string">
                     <pattern value="[_.A-Z]"/>
                  </restriction>
               </simpleType>
            </attribute>
         </extension>
      </simpleContent>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="0" name="simple_time" type="string"/>
<element default="H0H" maxOccurs="1" minOccurs="0" name="def_time" type="string"/>
<element maxOccurs="1" minOccurs="0" name="timerange">
   <complexType>
      <sequence>
         <element name="LowerBound" type="string"/>
         <element name="UpperBound" type="string"/>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="0" name="file1" type="string"/>
<element maxOccurs="1" minOccurs="0" name="data_source" type="string"/>
<element maxOccurs="1" minOccurs="0" name="data_source_item">
   <complexType>
      <sequence>
         <element maxOccurs="unbounded" name="DataSourceItem">
            <complexType>
               <sequence>
                  <element maxOccurs="1" minOccurs="1" name="Path" type="string"/>
                  <element maxOccurs="1" minOccurs="1" name="ItemName" type="string"/>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>
</element>
<element maxOccurs="1" minOccurs="0" name="data_library" type="string"/>
```

```
            <element maxOccurs="1" minOccurs="0" name="olap_member">
               <complexType>
                  <sequence>
                     <element maxOccurs="unbounded" name="OlapMember">
                        <complexType>
                           <sequence>
                              <element maxOccurs="1" minOccurs="1" name="UniqueName" type="string"/>
                              <element maxOccurs="1" minOccurs="0" name="Label" type="string"/>
                           </sequence>
                        </complexType>
                     </element>
                  </sequence>
               </complexType>
            </element>
         </sequence>
      </complexType>
      <element name="stpAllParm1Response">
         <complexType>
            <sequence>
               <element name="stpAllParm1Result">
                  <complexType>
                     <sequence>
                        <element maxOccurs="1" minOccurs="0" name="Streams">
                           <complexType>
                              <sequence>
                                 <element maxOccurs="1" minOccurs="0" name="_WEBOUT">
                                    <complexType>
                                       <sequence>
                                          <element name="Value" type="base64Binary"/>
                                       </sequence>
                                       <attribute name="contentType" type="string"/>
                                    </complexType>
                                 </element>
                              </sequence>
                           </complexType>
                        </element>
                     </sequence>
                  </complexType>
               </element>
            </sequence>
         </complexType>
      </element>
   </schema>
</types>
```

Index

Your Turn