SAS Publishing

SAS/IntrNet® 9.1 SAS/CONNECT® Driver for Java

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *SAS/IntrNet® 9.1: SAS/CONNECT® Driver for Java*. Cary, NC: SAS Institute Inc.

SAS/IntrNet® 9.1: SAS/CONNECT® Driver for Java

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

Published July 2002, July 2004, December 2004

Table of Contents

About the Tools

As the Java programming language continues to evolve, more and more Internet programmers embrace it for creating Web applications and thin clients. Our Java tools enable you to create portable Java programs that leverage the power of SAS software.

The SAS/CONNECT driver for Java enables you to write Java programs that take advantage of the computing and processing power of SAS software, as well as data access. With this driver, you can create Java programs that start and connect to a SAS session, access SAS data and run procedures to analyze the data, and retrieve the results. The SAS/CONNECT driver for Java package also comes with the SAS/SHARE driver for JDBC, which enables you to create Java programs that let users access and update data through a direct connection to a SAS/SHARE server.

We also offer the tunnel feature, which enables you to move your SAS server software to a machine other than your Web server and provides a way to navigate firewalls.

Debugging Tips

This page provides debugging tips for some of the problems you might encounter while trying to run an applet. For more troubleshooting information, see our troubleshooting page at **`support.sas.com/rnd/web/intrnet/misc/support.html`**.

Create a stable environment.

When you are experiencing problems with an applet, the first thing you should do is create a stable environment. For our Java tools, we recommend that you use the Java Plug–in or Appletviewer (both are available from JavaSoft). You can download the Java Plug–in from the JavaSoft site at java.sun.com/products/plugin.

Once you install the Java Plug–in or Appletviewer, run the applet that is not functioning properly. If it behaves as expected in this environment, the problem most likely is caused by an incompatibility with the applet and your browser. If the applet still does not function, check the exceptions and messages that are returned, then look for more information in the remainder of this guide.

If you are running the Java Plug–in, you may need to turn on the Java console:

1. Close your Web browser.
2. Select **Java Plug–In ControlPanel** from your Programs list.
3. Select **Show Java Console** from the Properties window.
4. Select **Apply**.

The next time you run an applet, the Java console will display. In addition to reviewing the information provided here, you should also review the information provided in the FAQ for the plug–in at java.sun.com/products/plugin/plugin.faq.html.

If you are using the tunnel feature, turn on logging.

The tunnel feature includes a logging mechanism that is an excellent debugging tool. If you are having problems running tunneled applets, add `LOG=`*`filename`* to the configuration file for the tunnel feature to turn logging on.

Note: This file can get large. Be sure to periodically delete or move the contents of this file.

For more information, refer to troubleshooting information for tunneled applets.

Verify your installation and system requirements.

Review the instructions (especially the sections on testing your installation) provided in the readme.txt files. You may also want to verify your directory structure by referring to the information in the archive.txt or package.txt files.

If the applet is installed correctly, verify that your environment meets the requirements for that applet or driver. View the requirements of the appropriate Java tool now:

- requirements for SAS/CONNECT driver for Java
- requirements for the tunnel feature.

About the SAS/CONNECT Driver for Java

The SAS/CONNECT driver for Java provides the Java classes that enable you to write Java programs that communicate with a SAS/CONNECT server. It enables you to take advantage of the computational capabilities of SAS from within your Java applications, applets, and servlets. (While the SAS/CONNECT driver for Java classes can be used to create Java applets, applications, and servlets, this documentation focuses primarily on *applet* development.)

Using the SAS/CONNECT driver for Java, you can create Java applications or applets that start a SAS session, connect to that session, create data sets, access existing SAS data, run procedures to analyze SAS data, and retrieve the results. The SAS/CONNECT driver for Java enables you to take advantage of remote computing resources on the SAS server machine. These remote capabilities turn your Java applications and applets into thin−client Web applications.

In addition to enabling you to submit SAS statements, the SAS/CONNECT driver for Java classes also provide a way to submit SQL statements, so you can access and update SAS data from your SAS/CONNECT driver for Java applets. You should also investigate using the tunnel feature that is available for use with the SAS/IntrNet Java components.

SAS/CONNECT Driver for Java Overview

The SAS/CONNECT driver for Java provides the API that enables you to write Java programs that access the compute services of SAS software. It accesses SAS software by communicating with a SAS/CONNECT server. The SAS/CONNECT driver for Java classes provide a subset of the functionality provided by SAS/CONNECT software.

The SAS/CONNECT driver for Java provides the classes necessary to write your own Java applets that

- use processing resources that are remote to the Web browser running the applet
- run SAS programs on a SAS server and return information and output to the browser running the Java applets.

The SAS/CONNECT driver for Java sample applet demonstrates some of the capabilities that you can write into your own Java programs using SAS/CONNECT driver for Java classes. You can find this applet in the SAMPLE directory of the Java Tools archive. The readme file contains instructions about how to configure this applet to work at your site.

Requirements for the SAS/CONNECT Driver for Java

If you are developing applets or applications using the SAS/CONNECT driver for Java, your development environment must support the Java Development Kit (JDK), version 1.4.1. An earlier version of the JDK might work but is not supported by SAS.

The SAS/CONNECT driver for Java contains the SAS/SHARE driver for JDBC, which is compliant with the JDBC 2.0 API and requires JDK 1.4.1 or later.

In order to test the functionality of your applets and applications, your development environment must also meet the requirements outlined in "Requirements for Deploying Applets."

The applets and applications you write using SAS/CONNECT driver for Java can communicate with a SAS server that is running SAS, Version 6 or later. The requirements for the SAS server vary based on the version of SAS software that is installed. Be sure to read the requirements carefully.

Requirements for Deploying Applets

After you have developed your applets and are ready to make them available to users, make sure that your intranet or internet meets the following requirements.

Web Server

Install the SAS Java archive on your Web server and move the applets you developed to the necessary directory.

Note: If you are not using the tunnel feature or the Java Plug–In, your Web server must be installed on the same physical machine as your SAS server.

SAS Server

- **SAS server running SAS, Version 6**

  Your SAS server must have Release 6.12 (TS050) or later of SAS installed. Your SAS installation must include at least the following SAS products:

  - SAS/CONNECT software
  - SAS/IntrNet software license
  - If your applications rely on any other products or tools provided by SAS, you must also install those products on your SAS server.

  The SAS server must also have either the SAS spawner program or a telnet daemon available. If you choose to use the spawner program, you must use the version provided with the SAS/IntrNet server. (This is a modified version of the SAS spawner program. It does not replace the version you currently have installed. The filename for the version provided in the SAS/IntrNet server package is tspawner.exe for Windows platforms and is sastcpdt for UNIX platforms.)

  Note: Currently, the modified version of the SAS spawner program is not available for the SunOS 4 platform.
- **SAS server running SAS, Version 7 or later**

Your SAS server must have SAS, Version 7 or later installed. Your SAS installation must include at least the following SAS products:

- SAS/CONNECT software
- SAS/IntrNet software license
- If you want to enable encrypted communication between the client and server, you must also install SAS/SECURE software.
- If your applications rely on any other products or tools provided by SAS, you must also install those products on your SAS server.

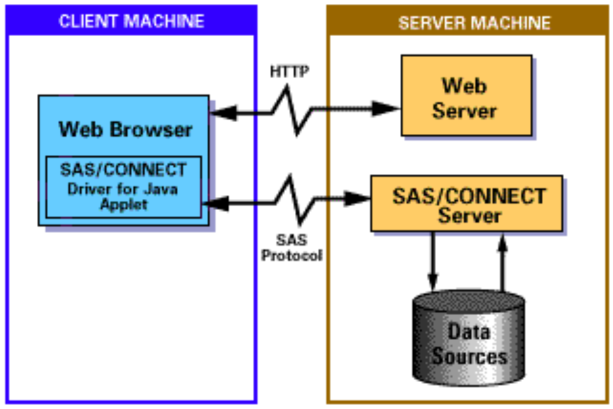
The SAS server must also have either the SAS spawner program or a telnet daemon available. The spawner program is provided with SAS/CONNECT software.

How SAS/CONNECT Driver for Java Works

With the SAS/CONNECT driver for Java classes, you can interactively submit SAS statements and retrieve log and output information. A Java program uses the SAS/CONNECT driver for Java classes to start a SAS/CONNECT session on the server machine and establish a connection to that session. Each program creates a SAS session for its use. SAS/CONNECT sessions are not shared between multiple users or multiple programs. The SAS session is destroyed when the program completes. After the program has established a connection to the remote SAS session, the applet can submit SAS statements and retrieve the log lines and output generated from those statements.

The SAS/CONNECT driver for Java provides classes that use either socket–based communication or the tunnel feature. Socket communication with applets can be restricted by the Java security manager. The security manager is provided by the browser, and each browser has a different set of restrictions imposed on applets. Most security managers restrict socket–based communication and only allow applets to open socket connections to the same machine that provided the applet classes. This means the SAS/CONNECT session and the Web server that provides the Java classes must run on the same machine. The tunnel feature eliminates this restriction, enabling you to provide your Web server and your SAS/CONNECT session on different machines.

The following diagram shows how all the components work together when you are using a Java applet. The Web browser on the client machine requests an HTML document from the Web server. The server responds by sending the document to the browser. If the browser detects an applet tag in the document, it sends additional requests to the Web server for the Java classes used by the applet. After the classes are downloaded to the client machine, the applet begins running.



Connecting to a Remote SAS Session

A SAS/CONNECT driver for Java program communicates with the SAS/CONNECT server using a connection daemon (either the SAS spawner program or a telnet daemon). A *tunneled* SAS/CONNECT driver for Java program communicates with the tunnel feature's server programs, which in turn communicate with the SAS/CONNECT server using either the spawner program or a telnet daemon. This page describes the types of connection objects you can use, how to construct a connection object, and other topics related to managing the connection to a remote SAS server.

The SAS/CONNECT driver for Java class you use to construct the connection object depends on whether your program uses HTTP tunneling or the SAS/CONNECT protocol. For HTTP tunneling, you will use the TunneledConnectClient class to create the connection object. Otherwise, you will use the TelnetConnectClient class to create the connection object.

Connection Properties

Connection objects are constructed using a set of properties that specify information that is required by the connection daemon. The properties represent the prompts and responses that are passed when the connection is established. The connection object waits for certain prompts from the connection daemon (or from the tunnel feature's server programs) and then responds. The properties are set from the values in a Properties object that is passed as an argument to the constructor of the connection object class you use.

For TelnetConnectClient and TunneledConnectClient, you must specify at least one property. If the property list is null, an exception is returned. The connection properties that you can specify for TelnetConnectClient and TunneledConnectClient objects are as follows:

promptx
: Where *x* is any number, and the value corresponds to the prompt you want to pass.

responsex
: Where *x* is any number, and the value corresponds to the connection information you want to pass. If you want your program to prompt the user for a username and password at startup, do not specify values for the response*x* properties you define for username and password information.

userNameResponse
passwordResponse
: These properties specify which response*x* properties contain the username and password information. For example, if you use **`response1`** for the username and **`response2`** for the password, you would specify **`response1`** as the value for the userNameResponse property and **`response2`** as the value for the passwordResponse property.

  For non−tunneled programs, you need to use the userNameResponse and passwordResponse properties only if you want the program to prompt for the username and password. If you want to specify the username and password as values for response*x* properties instead of having the user enter the information, do not use the userNameResponse and passwordResponse properties.

  For tunneled programs, the userNameResponse and passwordResponse properties specify which responses are the username and password so the responses can be encrypted before they are sent to the tunnel feature's server programs. If you do not use the userNameResponse and passwordResponse properties, the username and password will not be encrypted when they are sent to the server.

sasPortTag
: The tag in the message from SAS software indicating the port at which the SAS server is listening.

The prompt/response pairs required for a successful connection may differ depending on the connection daemon you are using. Typically, you need only create three prompt/response pairs to pass the username, password, and command information, but you can create additional parameters to pass prompts and responses that your connection daemon requires. For further explanation of how the prompt/response properties relate to the connection information that is passed when manually establishing a connection, see Connection Properties and Telnet Connection Information.

The Tunneling Property

Connection objects that use HTTP tunneling use the routerUrl property, which enables the Web server to locate the Message Router (`shrcgi.exe`), one of the tunnel feature's server programs.

Note: The routerUrl property applies to TunneledConnectClient objects only.

routerUrl
: The URL of the Message Router (`shrcgi.exe`).

Timeout Properties

To determine whether the connection process is continuing properly, the connection client allows a certain amount of time to pass waiting for a particular prompt. If the prompt is not received within the allotted time, the connection client assumes an error has occurred and throws a ConnectException. The connection client supports properties that enable a user to override the default timeout values. You can create as many Timeout properties as you need.

Note: Timeout properties apply only to TelnetConnectClient objects.

promptTimeoutx
: Where *x* is a number that corresponds to the prompt you want to time. You will specify a number of seconds you want the connection client to wait from the time it begins listening for the prompt to the time it receives the prompt.

responseTimeoutx
: Where *x* is a number that corresponds to the response you want to time. You will specify the number of seconds you want the connection client to wait from the time it sends the response to the time the connection daemon receives the response.

sasPortTagTimeout
: The sasPortTagTimeout property specifies the number of seconds you want the connection client to wait from the time it begins listening for the port tag to the time it receives the port tag.

Encryption Properties

When communicating with a SAS server that is running the SAS System, Version 7 or later, messages passed to and received from both the SAS Spawner and the SAS/CONNECT Server can be encrypted using a variety of encryption algorithms. The tunnel feature server programs do not support this level of encryption. The only supported encryption in the tunnel server programs is to encrypt username and password.

Two properties control most of the encryption features in the SAS/CONNECT Driver for Java.

encryptionPolicy
: This property controls whether or not the connection client will attempt to negotiate and use an encryption algorithm with the server and what to do if the negotiations are not successful. If the value of this property is **`none`**, then the connection client will not attempt to negotiate and use an

encryption algorithm with the server. If the server requires encryption to be used, then the connection will fail. This is the default value. If the value of this property is **`optional`**, then the connection client will attempt to negotiate and use an encryption algorithm with the server. If the negotiations fail, then the connection client will try to continue with an unencrypted connection. However, that will also fail if the server requires encryption. If the value of this property is **`required`**, then the connection client will attempt to negotiate and use an encryption algorithm with the server. If the negotiations fail, then the connection fails.

encryptionAlgorithms

This property specifies a comma–separated list of encryption algorithms in order of preference. The connection client will use this list when negotiating encryption algorithms with the server. It is not necessary to list all the algorithms that the connection client can support, only the ones you want to use for a particular connection. Also, if no algorithms are listed, then the server will choose one. Possible values for this property are **`sasproprietary`**, a stream cipher developed at SAS Institute, **`RC2`**, a block cipher developed by RSA Data Security, **`RC4`**, a stream cipher developed by RSA Data Security, **`des`**, a block cipher known as Data Encryption Standard, and **`TRIPLEDES`**, DES applied three times with separate keys. To use **`RC2`**, **`RC4`**, **`DES`**, or **`TRIPLEDES`**, you must license SAS/SECURE on the server and install the Java component of SAS/SECURE (sasecjav.zip) in your code base. **`TRIPLEDES`** can generally only be used in the United States and Canada due to export restrictions.

Because encryption is supported for connections to both the SAS Spawner and the SAS/CONNECT server, a property is sometimes needed to determine whether or not the encryption properties listed above apply to the telnet session or the SAS/CONNECT session. This property is named encryptionTarget. Its possible values are:

telnet

encryption properties apply only to the telnet session.

sas

encryption properties apply only to the SAS/CONNECT session. This is the default.

both

encryption properties apply to both the telnet session and the SAS/CONNECT session.

If the SAS Spawner was started with the –INHERITANCE option, then, in effect, the only possible value for this property is **`both`**. Therefore, this property is ignored when the –INHERITANCE option is used.

Encryption Properties for the Server

The SAS Server and SAS Spawner both require similar properties to control encryption. These properties can be specified as command line options when starting the SAS Spawner or when specifying the response*x* property that corresponds to the SAS command for TelnetConnectClient.

–NETENCRYPT

Specify this option to make the SAS Server or SAS Spawner require encryption when clients connect. If you do not specify this option, then the SAS Server or SAS Spawner will consider encryption to be optional.

–NETENCRALG

Specify this option and follow it with a comma–separated list of encryption algorithms. The list specifies the order of preference for use in algorithm negotiation. However, if you specify a list of encryption algorithms as a value to the encryptionAlgorithms property for the connection client, that list will take precedence. If this option is not specified, then encryption may not function properly. The values that you can use in the list of encryption algorithms are the same as the values you can use for the encryptionAlgorithms property for the connection client.

Other Useful Properties

You may also want to use the following properties.

compressionPolicy

Specify this option to manipulate compression of data in messages exchanged with the server. Possible values are:

session

specifies that all messages exchanged in a session are examined and compressed if the compression would yield a smaller message. This is the default behavior.

none

specifies that no messages should be compressed. This behavior is useful in SAS/CONNECT sessions in which the time used to compress and decompress data does not justify the amount of space saved by compression. This may be the case when the data exchanged is primarily binary.

textTransportFormat

Specifies the name of the character encoding used when moving text between the SAS/CONNECT Driver for Java and a SAS/CONNECT server. When you set this property, the driver transcodes SAS programs to this encoding before submitting them to the server. Transcoding from the specified encoding is automatically applied to log lines and list lines when they are received from the server. Note that downloaded files are not automatically transcoded when they are received from the server because the SAS/CONNECT driver for Java does not distinguish between text and binary files. However, when you download a text file, you can transcode it using any transcoding mechanism in the JDK along with the name of the encoding. You can fetch the name of the encoding using the getTextTransportFormat method.

If you do not specify a value for this property, the default character encoding of the Java virtual machine is used. If the specified character encoding cannot be supported by the Java virtual machine, then the SAS/CONNECT Driver for Java throws an exception the first time it attempts a character conversion.

Notes:

◊ Support for the Cp1047 code page is built into the SAS/CONNECT driver for Java because many current Java virtual machines do not provide support for this common character encoding.
◊ If you use the textTransportFormat property, you may need to reconfigure your SAS/CONNECT server as well.
◊ The SAS/SHARE driver for JDBC, which is included with the SAS/CONNECT driver for Java, transcodes character data, column names, column descriptions, and format names from this encoding when processing an SQL query.

logException

Specify this option so that your program will throw an exception when an error line appears in the SAS log. Possible values are **`TRUE`** and **`FALSE`**. The default behavior is FALSE, which does not attempt to determine if an error condition exists.

Using Applet Parameters to Pass Connection Information

An applet passes connection information through parameters, using the following format:

```
<param name=promptx value="prompt">
```

```
<param name=responsex value="response">
<param name=userNameResponse value="response1">
<param name=passwordResponse value="response2">
<param name=sasPortTag value="port_tag">
```

Where *x* is any number, and **`prompt`** and **`response`** are the values associated with the prompt and connection information you want to pass, respectively. The userNameResponse and passwordResponse parameters are as described above. **`port_tag`** uniquely identifies that the SAS session has completed initialization and allows the connection client to parse the port value from the response.

Note: You will use the getParameter method to pick up these values. Then, you construct a new Properties object and `put` these values into the object.

Tunneling information is passed through a parameter using the following format:

```
<param name=routerURLx value="http://your_server/cgi-bin/shrcgi">
```

Where *http://your_server/cgi−bin/shrcgi* is a URL that corresponds to the location of the tunnel feature's server programs.

Timeout information is passed through parameters, using the following format:

```
<param name=promptTimeoutx value="nnn">
<param name=responseTimeoutx value="nnn">
<param name=sasPortTagTimeout value="nnn">
```

Where *x* is a number that corresponds to the prompt or response you want to time, and *nnn* is the number of seconds you want the connection client to wait.

Constructing the Connection Object

If you want to use SAS/CONNECT protocol, instantiate the connection object using the TelnetConnectClient constructor, as follows:

```
import com.sas.net.connect.TelnetConnectClient;
TelnetConnectClient tconnection = new TelnetConnectClient(info);
```

If you want to use HTTP tunneling, instantiate the connection object using the TunneledConnectClient constructor as follows:

```
import com.sas.net.connect.TunneledConnectClient;
info.put("routerUrl","http://your_server/cgi-bin/shrcgi");
TunneledConnectClient tconnection = new TunneledConnectClient(info);
```

The TunneledConnectClient sends a request to the tunnel feature's server programs to start a SAS/CONNECT session and return the port number for communication with the Java client. Then, it passes connection properties to the Message Router.

Ending the Remote SAS Session

The session ends differently depending on whether your using SAS/CONNECT protocol or HTTP tunneling.

If you're using SAS/CONNECT protocol, the connection client class (TelnetConnectClient) overrides the base class disconnect method, so it can destroy the telnet or spawner session in addition to ending the remote SAS session. The class first calls the base class disconnect method, which sends a shutdown message to SAS and receives a response that the SAS session has successfully shut down. The subclass disconnect method then closes the socket connection to the telnet or spawner session, which effectively ends the telnet or spawner session. The telnet or spawner session cannot be destroyed prior to ending the SAS session. The SAS session is started as a subprocess of the telnet or spawner session, and ending the telnet or spawner session prematurely will end the SAS session.

If you're using HTTP tunneling, the base class disconnect method requests that the tunnel feature's server programs end the remote SAS session. When the base class disconnect method is called, a shutdown message is sent to the SAS session and a response is returned indicating that the SAS session has successfully shut down. Then, the tunnel feature's server programs end the telnet or spawner session. The telnet or spawner session cannot be destroyed prior to ending the SAS session. The SAS session is started as a subprocess of the telnet or spawner session, and ending the telnet or spawner session prematurely will end the SAS session.

SAS/CONNECT Server Configuration and the textTransportFormat Property

The textTransportFormat property specifies the text transport format (that is, the encoding of character data exchanged between the SAS/CONNECT driver for Java and the SAS/CONNECT server). This page documents the way to specify the textTransportFormat property and configure the SAS/CONNECT server so that the driver and server agree on the text transport format.

Note: For SAS/CONNECT 8.2 and later servers, the textTransportFormat property is not used because the value of textTransportFormat is automatically determined when connecting to those releases of the SAS/CONNECT server.

By default, the SAS/CONNECT server assumes that the transport format for text should be ASCII. Typically, that default is only useful when:

- the client program using the SAS/CONNECT driver for Java is running on an ASCII−based machine such as a PC or UNIX workstation
- the language used by the client program is English.

If either of those conditions are not in effect, then configuring the SAS/CONNECT driver for Java and the SAS/CONNECT server is necessary.

The best strategy to use when you find that the default text transport format needs to be changed is to make the text transport format exactly the same as the native character encoding of the machine hosting the SAS/CONNECT server. This strategy prevents you from having to make character encoding conversions in the SAS/CONNECT server, and it allows you to take advantage of the rich set of character encoding converters that are included with most Java virtual machines. To implement this strategy, you must:

1. Override the SAS/CONNECT server's default character encoding conversion mechanism so that it makes no conversion.
2. Inform the SAS/CONNECT driver for Java of the server's native character encoding so that it can use the Java virtual machine's character encoding converters to make the proper conversion.

The SAS/CONNECT server's character encoding conversion mechanism is controlled by a set of tables that translate character codes from one encoding to another. These tables, called *translation tables*, can be created and modified using the TRANTAB procedure and can be installed using the system option TRANTAB. To make the text transport format the same as the native character encoding of the machine hosting the SAS/CONNECT server, you need to create a translation table that "translates" each character code to itself.

The following SAS program creates such a translation table and stores it in your SASUSER.PROFILE catalog. Your SAS system administrator can copy it to the SASHELP.HOST catalog to make it available to all SAS users at your site.

```
proc trantab table = identity;
     replace "00"x "000102030405060708090A0B0C0D0E0F"x;
     replace "10"x "101112131415161718191A1B1C1D1E1F"x;
     replace "20"x "202122232425262728292A2B2C2D2E2F"x;
     replace "30"x "303132333435363738393A3B3C3D3E3F"x;
     replace "40"x "404142434445464748494A4B4C4D4E4F"x;
     replace "50"x "505152535455565758595A5B5C5D5E5F"x;
     replace "60"x "606162636465666768696A6B6C6D6E6F"x;
     replace "70"x "707172737475767778797A7B7C7D7E7F"x;
     replace "80"x "808182838485868788898A8B8C8D8E8F"x;
```

```
        replace "90"x "909192939495969798999A9B9C9D9E9F"x;
        replace "A0"x "A0A1A2A3A4A5A6A7A8A9AAABACADAEAF"x;
        replace "B0"x "B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF"x;
        replace "C0"x "C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF"x;
        replace "D0"x "D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF"x;
        replace "E0"x "E0E1E2E3E4E5E6E7E8E9EAEBECEDEEEF"x;
        replace "F0"x "F0F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF"x;
        inverse;
        save both;
quit;
```

After you have created the appropriate translation table, you can use the TRANTAB system option to read the translation table out of SASUSER.PROFILE or SASHELP.HOST and install it into the SAS/CONNECT server's character encoding conversion mechanism. The system option TRANTAB controls the mechanism for converting both from transport format to the native encoding of the server and from the native encoding of the server to transport format. Use the *identity* translation table for both purposes.

You can use any of a variety of ways to specify system options to SAS software. The example later in this section shows how to use the TRANTAB system option on the command line for UNIX and PC hosts. Consult SAS system options documentation if you want to see other ways of using it.

To inform the SAS/CONNECT driver for Java of the server's native character encoding, you use the name of the encoding as the value of the textTransportFormat option. Some character encodings have more than one commonly used name, so you need to be sure to choose the name for each encoding that is recognized by a Java virtual machine. JavaSoft maintains a list of recognized encoding names on their Web site (java.sun.com/products).

Example

The following example shows the applet parameters for the ConnectApplet that you can use to connect to a SAS/CONNECT server running on a PC with the SAS spawner in the United States or Western Europe. The important changes to note are the use of the textTransportFormat property and the use of the TRANTAB system option in the SAS command, which is specified in the value of the response3 property.

```
<param name=prompt1 value="Username:">
<param name=response1 value="">
<param name=prompt2 value="Password:">
<param name=response2 value="">
<param name=userNameResponse value="response1">
<param name=passwordResponse value="response2">
<param name=prompt3 value="Hello">
<param name=response3 value="sas -trantab '(identity,identity)'">
<param name=textTransportFormat value="Cp1252">
```

Connection Properties and Telnet Connection Information

The property values used by a SAS/CONNECT driver for Java program in connecting to a telnet daemon (or the spawner) map directly to the telnet connection information that is passed when manually establishing a remote SAS session via telnet. This page describes the relationship between the prompt/response pairs you define as connection properties and the telnet connection information that is passed when manually establishing a connection.

Note: In the case of SAS/CONNECT driver for Java *applets*, the connection property values are specified through the use of connection *parameters* in the applet HTML file.

Telnet Example

To better explain how the SAS/CONNECT driver for Java properties are used to start a remote SAS session via telnet, let's look at what happens when this operation is done manually.

The user wants to run SAS/CONNECT on a remote host named myhost. Myhost has a telnet daemon running and listening on a port. If the user telnets to myhost, the sequence of requests from myhost and responses from the user might look like the following:

```
Login: myuserid
Password: mypassword

There may be additional textual information sent from the
remote telnet session indicating the user has logged on.

Hostname> sas -dmr -noterminal -nosyntaxcheck

SAS(R) TCPIP REMOTE LINK PORT=1763          SESSION ESTABLISHED.
```

The telnet daemon provides the prompt for the user to log in. In this example, the login prompt is "`Login:`". The user responds with "myuserid". The password prompt is "`Password:`" and the user response is "mypassword". The command prompt is "`Hostname>`", and the user response is the command to start the SAS/CONNECT session: `sas -dmr -noterminal -nosyntaxcheck`.

The SAS session responds with a message that states which port it will use for communication with the client, "`PORT=`", followed by the port number (in this case, port number 1763).

Mapping Connection Properties to Telnet Connection Information

Just as the user receives certain prompts from the telnet session and responds to them, the connection client (specifically, TelnetConnectClient or TunneledConnectClient) behaves the same way. It waits for certain prompts from the telnet or spawner session (or from the tunnel feature's server programs) and then responds.

The prompt/response pairs you need depends on the connection daemon you are using. You can create all the prompt and response properties required to pass the information that your connection daemon requires, using the following format:

promptx

Where *x* is any number, and the value corresponds to the prompt you want to pass.

responsex

Where *x* is any number, and the value corresponds to the connection information you want to pass.

It is important that you label prompt/response pairs in numerical order, without skipping any numbers. For example, if you create a prompt5/response5 pair, but you do not define a prompt4/response4 pair, the prompt5/response5 properties will be ignored and the connection will fail.

Note: All but one of the connection properties is created using this format. One property, sasPortTag, is hardcoded to specify the port tag.

Extending the example provided above, the following illustrates how to specify the required connection information using three prompt/response pairs. In this case, we are using use connection properties to meet the most common requirements:

- The username prompt is specified using the prompt1 property; the username response is specified using the `response1` property.
- The password prompt is specified using the prompt2 property; the password response is specified using the response2 property.
- The command prompt is specified using the prompt3 property; the command response is specified using the response3 property.

The prompt/response pairs needed for this example are as follows:

(prompt1, Login:)

The connection client uses this value to determine when it should send the userName response.

Note: The prompt1 value is a *substring* that the connection client compares with the message from the telnet session. It must be a *unique* substring. If the complete telnet message is "`Login:`", the value "`in:`" would be a valid prompt1 value because it uniquely identifies the login prompt. Please note that "`:`" would not be a valid prompt1 value because it is not a unique identifier. The connection client would not know what information to prompt the user for, and the connection to the remote system would fail.

(response1, myuserid)

The connection client will send this response after it has received the username prompt from the telnet session.

(prompt2, Password:)

The connection client uses this value to determine when it should send the password response.

Note: The prompt2 value is a *substring* that the connection client compares with the message from the telnet session. It must be a *unique* substring. If the complete telnet message is "`Password:`", the value "`word:`" would be a valid prompt2 value, because it uniquely identifies the password prompt. Please note that ":" would not be a valid prompt2 value because it is not a unique identifier. The connection client would not know what information to prompt the user for, and the connection to the remote system would fail.

(response2, mypassword)

The connection client will send this response after it has received the password prompt from the telnet session.

(prompt3, Hostname>)

The connection client uses this value to determine when it should send the command.

Note: The prompt3 value is a *substring* that the connection client compares with the message from the telnet session. It must be a *unique* substring. If the complete telnet message is "`Hostname>`", the value ">" would be a valid prompt3value because it uniquely identifies the command prompt.

(response3, sas −dmr −noterminal −nosyntaxcheck)

This is the complete response to the command prompt.

(sasPortTag, PORT=)

This uniquely identifies that the SAS session has completed initialization and allows the connection client to parse the port value from the message.

Note: The sasPortTag value must be the substring that immediately precedes the port value. While the substring "ESTABLISHED" uniquely identifies the message, it does not immediately precede the port number and the connection client will fail to establish a connection to the SAS session.

Class Documentation

The class documentation for the SAS/CONNECT driver for Java is generated using the Javadoc tool. The documentation is installed with your software. To view the class documentation from the SAS Web site, go to support.sas.com/rnd/web/intrnet/java/jconnect.

About the Tunnel Feature

The tunnel feature is an optional feature that you can use with Java applets written using the SAS/SHARE driver for JDBC or the SAS/CONNECT driver for Java. It addresses two common configuration problems encountered with Java applets that communicate with another machine:

- A Java applet that is downloaded from a Web server is not allowed to make socket connections to machines other than the machine from which it was downloaded. This restriction means that the SAS session would have to be started on the same machine as the Web server from which the applets were downloaded.
- Many firewalls prohibit applets from establishing socket connections beyond the firewall. However, most firewalls allow HTTP protocol to pass through the firewall, so if the communication is done with HTTP protocol, applets are able to communicate with servers that they would not normally be allowed to communicate with.

Note: Java Plug–in software, from Sun Microsystems, Inc., allows you to configure the way your applets access machines on your network. If you are using the tunnel feature to allow an applet to communicate with machines other than the Web server from which it is downloaded, you can use the Java Plug–in software instead.

The tunnel feature can solve both of these problems by virtually eliminating the restrictions on where your SAS software runs in relation to your Web server and firewall. What's more, because the tunnel feature gives you complete control over who can access what, you gain these benefits without sacrificing security for your vital data. The tunnel feature enables you to greatly enhance the power and flexibility of the applets you create with the SAS/SHARE driver for JDBC and the SAS/CONNECT driver for Java.

The Tunnel Feature Overview

The security model currently used by most Web browsers restricts a Java client to communicating only with the Web server where the Java classes reside. This restriction requires that your SAS server be installed on the same machine as your Web server. For most enterprises, this is not an optimal configuration. The tunnel feature that we are providing lifts this restriction.

The tunnel feature consists of programs that are installed on your Web server. These programs, called the tunnel feature's server programs, use the Common Gateway Interface (CGI) to receive requests from the Java applet running on a user's browser. The server programs forward the requests to the SAS server for processing. When the processing is complete, the programs return the results to the applet.

Requirements for the Tunnel Feature

The tunnel feature works with the SAS/SHARE driver for JDBC and the SAS/CONNECT driver for Java. The tunnel feature's server programs are installed on a Web server. The Web server must have

- a directory in which CGI programs can be stored. This directory is often named cgi−bin on UNIX platforms and scripts on PC platforms. You will install the tunnel feature's server programs in this directory.
- access to a SAS/SHARE or SAS/CONNECT server.
- the SAS/SHARE driver for JDBC and the SAS/CONNECT driver for Java installed locally.

  Note: During installation of either driver, both drivers are automatically installed.

Before you provide programs that use the tunnel feature, be sure that the requirements for the SAS/SHARE driver for JDBC or for the SAS/CONNECT driver for Java are also met.

Supported Platforms

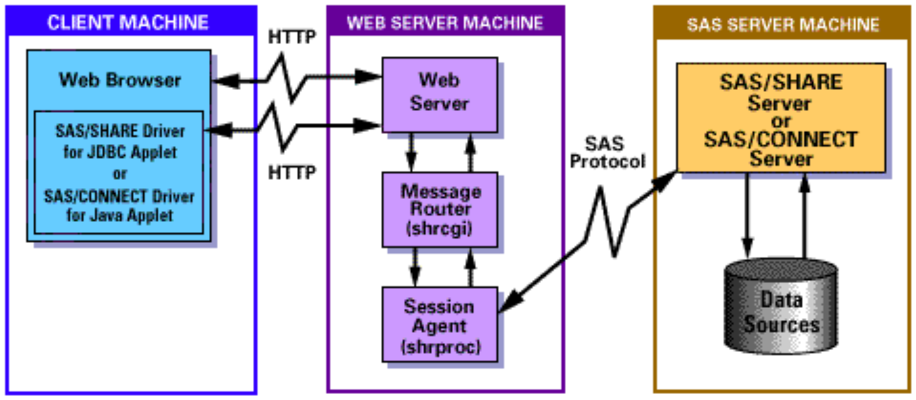
The tunnel feature is available for use on the following platforms:

- AIX/6000
- Compaq Tru64 UNIX
- HP−UX
- Linux
- Solaris
- Windows NT
- Windows 2000
- Windows 2003 Server
- Windows XP

How the Tunnel Feature Works

A common problem in deploying Java applets is the configuration limitations that are imposed when applets must communicate with machines other than the Web server from which they are downloaded. The tunnel feature eliminates this problem by employing HTTP (Hypertext Transfer Protocol) Tunneling to allow applets to communicate with remote systems via a CGI program running on the Web server.

The following figure illustrates how a request (a SAS statement or SQL statement) is sent from a Java applet to a SAS/CONNECT or SAS/SHARE server when you use the tunnel feature.



Initially, the Web browser (on the client machine) loads the applet HTML file from the Web server, which causes the required Java classes to be downloaded from the Web server as well. The Java classes (the SAS/CONNECT driver for Java, the SAS/SHARE driver for JDBC, or both) then communicate directly with the Web server (via HTTP) to pass a request from the applet.

Once the Web server has received the request, it passes it to the Message Router (shrcgi), one of the tunnel feature's server programs. The Message Router is a CGI program that will pass the request on to the SAS server, *provided the statement or request meets certain criteria*. By checking the tunnel feature's configuration file (which is created by the system administrator), the Message Router can determine if the request is

- coming from an approved client machine
- going to an approved SAS server machine, through an approved port
- coming from an approved user, with an approved level of access (for SAS/CONNECT driver for Java applets only)

Once the Message Router has determined that the statement or request is acceptable, it creates a detached process called the Session Agent (shrproc), which communicates with the SAS server machine. Based on the statement or request from the applet, the Session Agent either starts a SAS/CONNECT session on the SAS server machine or establishes a connection to a SAS/SHARE server. Then, the Message Router passes the statement or request to the Session Agent, and the Session Agent passes it directly to the SAS/CONNECT or SAS/SHARE server.

After the SAS server has processed the statement or request, it returns the data to the Session Agent. The Session Agent passes the data to the Message Router, which then passes it to the applet running on the client machine.

The Tunnel Feature's Configuration File

The information in the tunnel feature's configuration file controls access to remote SAS sessions. The configuration file is required; it allows the administrator to limit which machines are accessed and through which ports and client machines they are accessed.

For the SAS/CONNECT driver for Java, the configuration file also controls which user IDs are allowed to connect to the SAS server and the commands that those users are allowed to use to start the remote SAS session. Without the configuration file, there would be no limits on the commands that users could execute during the process of starting SAS.

Using a Configuration File

To use a configuration file, the administrator creates the shrcgi.cfg file, specifies any options that are needed, and stores it on the Web server in the directory specified by the environment variable, SHRCGI_CFG. If you have not set `SHRCGI_CFG`, the Message Router looks for the configuration file in your CGI directory.

Note: The Message Router reads the configuration file each time the communication session is started with a particular host.

The shrcgi.cfg file included with the tunnel feature archive is a configuration file *template*. You must modify it to specify configuration information that is specific to your site. Do not attempt to use the template file as the final configuration file because it does not include your specific host and port names.

A Sample Configuration File

```
# A Sample Configuration File

# A configuration file is required for the tunnel feature's server programs.
# This configuration file must be modified for your configuration. The tunnel
# feature will not work on your system if you have not updated this file with
# the proper information.

# The tunnel feature's server programs will look for this file in the same
# directory where they are located. You have the option of overriding this
# default using an environment variable. This option is documented in our
# documentation package.

# The following lines specify which hosts, ports, SAS commands and usernames
# are allowed by the tunnel feature's programs.

# Any lines preceding the first SASHOST line are considered global parameters
# and are added to all SASHOST groups below. The following TIMEOUT parameter
# would apply to all hosts.

TIMEOUT=60

# The SASHOST parameter in this config file specifies that the first host
# YOURTESTHOST will be allowed for SAS/CONNECT driver for Java applets and
# SAS/SHARE driver for JDBC applets.

SASHOST=YOURTESTHOST

# The SASPORT parameter specifies that the applets may connect to YOURTESTHOST
# on ports 23 and 5010. In this particular example, port 23 is the telnet port
# that is used to start the SAS/CONNECT session. Port 5010 is the port the
# SAS/SHARE server is listening on. No other ports are will be allowed
# access from the applets.

ALLOW_SASPORT=5010,23

# The following parameters are used by the SAS/CONNECT driver for Java only.
# They have  no meaning for JDBC. Only the command specified by the alias
# mySasCommand will be allowed for SAS/CONNECT driver for Java applets. You
# must change this command to a valid command to start SAS on the host
# YOURTESTHOST. All other commands will not be allowed.

mySasCommand = sas -dmr -noxcmd -nosyntaxcheck -noterminal -cleanup
ALLOW_RESPONSE_3=$mySasCommand

# Only user1 and user 2 may access host YOURTESTHOST.  All other users will be
# denied access.

ALLOW_USERNAME=user1,user2

# The next SASHOST parameter in this config file specifies that the other host
# SECONDTESTHOST will be allowed for SAS/CONNECT driver for Java applets and
# SAS/SHARE driver for JDBC applets.  No other hosts will be allowed access from
# the applets.

SASHOST=SECONDTESTHOST

# Any user except user3 may access host SECONDTESTHOST.

DISALLOW_USERNAME=user3
```

Configuration File Options

This page describes all the options you can specify in the tunnel feature's configuration file and provides some general guidelines for specifying the options.

Configuration File Guidelines

When you modify the shrcgi.cfg file, make sure you follow these guidelines:

- For each SASHOST, first specify the SASHOST identifier (which can include wildcards), then specify the options that apply to that host. If an option applies to more than one SASHOST, but not all SASHOSTs, you must repeat the option for each host. If you want an option to apply to all SASHOSTs, make it a *global option* by placing it before the first SASHOST line.
- The options SASPORT, RESPONSE_x, CLIENTHOST, and USERNAME have ALLOW and DISALLOW lists. The ALLOW lists and DISALLOW lists have similar functions: they both controls which machines and users are able to connect to the remote SAS sessions. Use one or the other depending on which will require a shorter list.

  The DISALLOW list takes precedence over the ALLOW list. When the tunnel feature's server programs receive a request from the applet, they check the DISALLOW list first. If the request matches any value in the DISALLOW list, the request is rejected. The ALLOW list is checked only if the DISALLOW list is not present or if the request did not match any values specified in the DISALLOW list. If an ALLOW list is present, the request must match an option in the ALLOW list; otherwise, the request is rejected.
- You can use the asterisk (*) wildcard when specifying values in ALLOW or DISALLOW lists. For example, C*AT matches CAT, CHAT, and CRAVAT.
- You can use aliases in the configuration file to mask the actual SAS command that is being used to invoke the remote SAS session. By masking the SAS command, the tunnel feature avoids exposing any specific information about the configuration of your system.
- You can specify only one set of options for each host. If you specify a second set of options for a host, the second set is ignored. For example, if you specify options for the host identifier TEST*, and then you specify options for TEST2, the tunnel feature ignores the options that are specified for TEST2. When it receives a request that includes TEST2, the tunnel feature checks the request against the options specified for TEST*.
- If a configuration file entry accepts multiple values, delimit the values with commas only.
- Leading spaces are ignored.
- Line continuation is not supported. You can use lines up to 256 characters.
- To enter a comment, enter a pound sign (#) as the first character on each line of the comment. The Message Router ignores lines that begin with the pound sign.

Configuration File Example

The following configuration options apply to two hosts: TESTER and WIZARD.

```
SASHOST=TESTER
ALLOW_USERNAME=XYZ,A*,QRS
DISALLOW_USERNAME=ABC

SASHOST=WIZARD
ALLOW_RESPONSE_3=sas,sas -dms
```

The USERNAME specifications apply only to the TESTER host, and the SASCOMMAND specification applies only to the WIZARD host. Only users with the IDs XYZ, QRS, and those starting with A, except ABC, can connect to the host machine TESTER. On WIZARD, the only SAS commands allowed are the two commands shown in the ALLOW list (assuming RESPONSE_3 is defined as the response to the command prompt).

Configuration Option

This section lists the options you can define in the tunnel feature's configuration file.

Options for SAS/SHARE Servers and SAS/CONNECT Servers

The following options apply to both SAS/SHARE and SAS/CONNECT:

SASHOST=hostname

Identifies the host (node) name or remote IP addresses of the machines on which your SAS/SHARE server is running or on which you want to start your SAS/CONNECT session. SASHOST specifies a single entry; it is not a comma delimited list. For each SASHOST, first specify the SASHOST identifier (which can include wildcards), then specify the options that apply to that host. Users cannot connect to hosts that are not included in the configuration file. To remove any restrictions on the hosts, use a wildcard to specify all hosts, `SASHOST=*`.

Note: The tunnel feature will look for an exact match, so if you specify a node name, but the request that tunnel feature receives uses the IP address for the same node, the tunnel feature will not recognize that the node name and IP address are for the same node.

ALLOW_SASPORT=port1,port2...
DISALLOW_SASPORT=port1,port2...

Lists the ports that can/cannot be used to establish a connection. For the SAS/CONNECT, specify the ports on which the telnet daemon or spawner will receive requests. For SAS/SHARE, list the public ports that SAS/SHARE server is listening to.

LOG=log_file_name

Identifies a log file that can be used for debugging tunneling problems.

Note: Information will be added to this log file every time the tunneling feature is used, potentially creating an extremely large file. Consider periodically deleting the contents of the file, or remove this option setting from the configuration file after your tunneling problems have been resolved.

ALLOW_RESPONSE_x=response1, response2...
DISALLOW_RESPONSE_x=response1, response2...

Lists the allowed/disallowed telnet (or spawner) responses where *x* is from 1 to 5. (The aliases $USERNAME and $PASSWORD are always allowed, provided you have defined them. See alias below.)

ALLOW_CLIENTHOST=node1,node2...
DISALLOW_CLIENTHOST=node1,node2...

Lists the node names or remote IP addresses of the machines that can/cannot connect to a SAS/SHARE server or start a SAS/CONNECT session. The tunnel feature will look for an exact match, so if you specify a node name, but the request that the tunnel feature receives uses the IP address for the same node, the tunnel feature will not recognize that the node name and IP address are for the same node. Also, remember that the apparent requester may be a proxy executing the HTTP request on behalf of another machine.

TIMEOUT=nnn

Specifies the amount of time (in minutes) that the tunnel feature is to wait for activity before closing the connection between the Protocol Interpreter and the SAS/SHARE server or SAS/CONNECT session. The default timeout is 30 minutes. After the timeout expires, the tunnel feature closes the Protocol Interpreter and the session ends.

WAIT=nnn

Specifies the amount of time (in seconds) that the tunnel feature is to wait when connecting to a SAS/SHARE server or SAS/CONNECT session. The default time is 60 seconds.

alias=response−to−substitute

Specifies a response to substitute for the alias. Aliases are case insensitive, and the first character must be a dollar sign ($). For example, if you define the following alias:

```
mycommand=sas -dmr
```

you could then refer to the command using its alias, like so:

```
ALLOW_RESPONSE=$mycommand
```

Options for SAS/SHARE Servers Only

The following option applies to SAS/SHARE only:

HELLO=nnn

Specifies the amount of time (in seconds) that the tunnel feature will wait for SAS/SHARE initialization processing, which occurs immediately after the connection to the SAS/SHARE server is established. The default time is 45 seconds.

Options for SAS/CONNECT Servers Only

The following option applies to SAS/CONNECT only:

ALLOW_USERNAME=user1,user2...
DISALLOW_USERNAME=user1,user2...

Lists the usernames that can/cannot be used to log in to the remote SAS session.

Using the Tunnel Feature

To use the tunnel feature, you must install the server programs and modify the configuration file to include your system information.

You must also ensure that the routerUrl property is passed to driver's communication classes. The routerUrl property is set as a parameter on the <applet> tag in your HTML file. If you are using the tunnel server programs, you should add a line to the <applet> tag that has the following format:

```
<param name=routerUrl value="http://yourhost.com/cgi-bin/shrcgi.exe">
```

Note: Your applet classes must be provided from the same Web server on which you have installed the tunnel feature's server programs. If the applet tries to access a different Web server, your applet will get a security violation.