SAS/CONNECT® 9.4 User's Guide, Fourth Edition

SAS® Documentation
January 22, 2024

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS/CONNECT® 9.4 User's Guide, Fourth Edition*. Cary, NC: SAS Institute Inc.

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

January 2024

9.4-P10:connref

Contents

Stylistic Conventions

Overview of Syntax Conventions for the SAS Language

SAS uses standard conventions in the documentation of syntax for SAS language elements. These conventions enable you to easily identify the components of SAS syntax. The conventions can be divided into these parts:

- syntax components
- style conventions
- special characters
- references to SAS libraries and external files

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=). The syntax for arguments has multiple forms in order to demonstrate the syntax of multiple arguments, with and without punctuation.

keyword
specifies the name of the SAS language element that you use when you write your program. Keyword is a literal that is usually the first word in the syntax. In a CALL routine, the first two words are keywords.

In these examples of SAS syntax, the keywords are bold:

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

In this example, the first two words of the CALL routine are the keywords:

CALL RANBIN(*seed*, *n*, *p*, *x*)

The syntax of some SAS statements consists of a single keyword without arguments:

DO;
... *SAS code* ...
END;

Some system options require that one of two keyword values be specified:

DUPLEX | **NODUPLEX**

Some procedure statements have multiple keywords throughout the statement syntax:

CREATE <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <, *column-2*, …>)

argument
specifies a numeric or character constant, variable, or expression. Arguments follow the keyword or an equal sign after the keyword. The arguments are used by SAS to process the language element. Arguments can be required or optional. In the syntax, optional arguments are enclosed in angle brackets (< >).

In this example, *string* and *position* follow the keyword CHAR. These arguments are required arguments for the CHAR function:

CHAR (*string, position*)

Each argument has a value. In this example of SAS code, the argument *string* has a value of 'summer', and the argument *position* has a value of 4:

```
x=char('summer', 4);
```

In this example, *string* and *substring* are required arguments, whereas *modifiers* and *startpos* are optional.

FIND(*string, substring* <, *modifiers*> <, *startpos*>

argument(s)
specifies that one argument is required and that multiple arguments are allowed. Separate arguments with a space. Punctuation, such as a comma (,) is not required between arguments.

The MISSING statement is an example of this form of multiple arguments:

MISSING *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2* ... >
specifies that one argument is required and that a literal argument can be associated with the argument. You can specify multiple literals and argument pairs. No punctuation is required between the literal and argument pairs. The ellipsis (...) indicates that additional literals and arguments are allowed.

The BY statement is an example of this argument:

BY <DESCENDING> *variable-1* <<DESCENDING> *variable-2* …>;

argument-1 <*options*> <*argument-2* <*options*> ...>
specifies that one argument is required and that one or more options can be associated with the argument. You can specify multiple arguments and associated options. No punctuation is required between the argument and the

option. The ellipsis (...) indicates that additional arguments with an associated option are allowed.

The FORMAT procedure PICTURE statement is an example of this form of multiple arguments:

PICTURE name <(*format-options*)>
<*value-range-set-1* <(*picture-1-options*)>
<*value-range-set-2* <(*picture-2-options*)> …>>;

argument-1=value-1 <argument-2=value-2 ...>
specifies that the argument must be assigned a value and that you can specify multiple arguments. The ellipsis (...) indicates that additional arguments are allowed. No punctuation is required between arguments.

The LABEL statement is an example of this form of multiple arguments:

LABEL *variable-1=label-1 <variable-2=label-2* …>;

argument-1 <, *argument-2*, ...>
specifies that one argument is required and that you can specify multiple arguments that are separated by a comma or other punctuation. The ellipsis (...) indicates a continuation of the arguments, separated by a comma. Both forms are used in the SAS documentation.

Here are examples of this form of multiple arguments:

AUTHPROVIDERDOMAIN (*provider-1:domain-1* <, *provider-2:domain-2*, …>

INTO :*macro-variable-specification-1* <, :*macro-variable-specification-2*, …>

Note: In most cases, example code in SAS documentation is written in lowercase with a monospace font. You can use uppercase, lowercase, or mixed case in the code that you write.

Style Conventions

The style conventions that are used in documenting SAS syntax include uppercase bold, uppercase, and italic:

UPPERCASE BOLD
identifies SAS keywords such as the names of functions or statements. In this example, the keyword ERROR is written in uppercase bold:

ERROR <*message*>;

UPPERCASE
identifies arguments that are literals.

In this example of the CMPMODEL= system option, the literals include BOTH, CATALOG, and XML:

CMPMODEL=BOTH | CATALOG | XML |

italic
identifies arguments or values that you supply. Items in italic represent user-supplied values that are either one of the following:

- nonliteral arguments. In this example of the LINK statement, the argument *label* is a user-supplied value and therefore appears in italic:

  LINK *label*;

- nonliteral values that are assigned to an argument.

  In this example of the FORMAT statement, the argument DEFAULT is assigned the variable *default-format*:

  FORMAT *variable(s)* <*format* > <DEFAULT = *default-format*>;

Special Characters

The syntax of SAS language elements can contain the following special characters:

=
an equal sign identifies a value for a literal in some language elements such as system options.

In this example of the MAPS system option, the equal sign sets the value of MAPS:

MAPS=*location-of-maps*

< >
angle brackets identify optional arguments. A required argument is not enclosed in angle brackets.

In this example of the CAT function, at least one item is required:

CAT (*item-1* <, *item-2*, …>)

|
a vertical bar indicates that you can choose one value from a group of values. Values that are separated by the vertical bar are mutually exclusive.

In this example of the CMPMODEL= system option, you can choose only one of the arguments:

CMPMODEL=BOTH | CATALOG | XML

...
an ellipsis indicates that the argument can be repeated. If an argument and the ellipsis are enclosed in angle brackets, then the argument is optional. The repeated argument must contain punctuation if it appears before or after the argument.

In this example of the CAT function, multiple *item* arguments are allowed, and they must be separated by a comma:

CAT (*item-1 <, item-2*, …>)

'*value*' or "*value*"
indicates that an argument that is enclosed in single or double quotation marks must have a value that is also enclosed in single or double quotation marks.

In this example of the FOOTNOTE statement, the argument *text* is enclosed in quotation marks:

FOOTNOTE <*n*> <*ods-format-options* '*text*' | "*text*">;

;
a semicolon indicates the end of a statement or CALL routine.

In this example, each statement ends with a semicolon:

```
data namegame;
   length color name $8;
   color = 'black';
   name = 'jack';
   game = trim(color) || name;
run;
```

References to SAS Libraries and External Files

Many SAS statements and other language elements refer to SAS libraries and external files. You can choose whether to make the reference through a logical name (a libref or fileref) or use the physical filename enclosed in quotation marks.

If you use a logical name, you typically have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the reference. Several methods of referring to SAS libraries and external files are available, and some of these methods depend on your operating environment.

In the examples that use external files, SAS documentation uses the italicized phrase *file-specification*. In the examples that use SAS libraries, SAS documentation uses the italicized phrase *SAS-library* enclosed in quotation marks:

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

What's New in SAS/CONNECT 9.4

Overview

The following features are new or enhanced for SAS/CONNECT 9.4:

- new CONNECTEVENTS system option
- enhanced RSUBMIT command with NEW statement option
- new XATTR= data set option in PROC UPLOAD and PROC DOWNLOAD
- new SAS/CONNECT spawner start-up options and new management interface
- enhanced logging and messaging
- enhanced data transfer of encoded data
- added flexibility for password and user ID naming
- enhanced password support on z/OS
- support for new Base SAS language elements
- support for extended attributes
- support for the new default values of the LRECL= Option

SAS/CONNECT 9.4M1 has the following enhancement:

- new locked-down state restrictions

SAS/CONNECT 9.4M2 has the following enhancement:

- enhanced INFILE option in PROC UPLOAD and PROC DOWNLOAD

SAS/CONNECT 9.4M3 has the following enhancement:

- document enhancements

SAS/CONNECT 9.4M5 has the following enhancements:

- new Authinfo file support for credentials
- new TCPPROXYLIST environment variable
- changed default value for the TCPLISTENTIME option
- new NOCLEARTEXT default spawner behavior

- new error message for sign-ons from workspace servers that allow numeric session-ids
- added _USER_ option to %SYSRPUT statement

New CONNECTEVENTS System Option

In SAS/CONNECT 9.4 the new CONNECTEVENTS system option specifies whether SAS events are propagated from the SAS/CONNECT server through the SAS/CONNECT client to SAS Enterprise Guide or to the Add-In for Microsoft Office (AMO). For more information, see “CONNECTEVENTS” on page 102.

Enhanced RSUBMIT Command with NEW Statement Option

In SAS/CONNECT 9.4 when you specify the LOG= statement or the OUTPUT= statement in the RSUBMIT command, you can now specify NEW to open the file for output and clear the log. Specifying NEW keeps your log file from appending, which is the default behavior for log files in SAS/CONNECT. If the log file or output file already exists, then it is deleted and re-created rather than appended. For more information, see LOG=.

New XATTR= Data Set Option in PROC UPLOAD and PROC DOWNLOAD

In SAS/CONNECT 9.4 the new XATTR= option in the PROC UPLOAD and PROC DOWNLOAD statements enables you to specify whether to transfer extended attributes with a SAS data set or SAS library.

New SAS/CONNECT Spawner Start-Up Options and New Management Interface

The SAS/CONNECT spawner features a new management interface that is compatible with the Windows, UNIX, and z/OS operating environments. The new interface enables administrators to monitor and manage the SAS/CONNECT spawner using SAS Management Console or PROC IOMOPERATE. A new spawner start-up command and new spawner options enable administrators to control how the spawner starts and operates.

Enhanced Logging and Messaging

- Improved Messaging: In grid-enabled sign-ons, you can now see the job ID and the grid host output in the log.
- SAS Logging Facility: The SAS logging facility is now the standard debugging tool for using SAS/CONNECT in a client/server environment. The Logging Facility offers new functionality, new appenders, and new loggers for monitoring SAS/CONNECT and providing more detailed debug tracing. For more information about using the Logging Facility in SAS/CONNECT, see Administering Logging for SAS/CONNECT .

Enhanced Data Transfer of Encoded Data

SAS 9.4 supports UTF-8 to Non-UTF-8 client/server connections. Connections can now be made between client/server sessions in which one session is using UTF-8 encoded data and the other session is using non-UTF-8 encoded data. For more information about client/server session compatibility, see Encoding Compatibility between SAS/CONNECT Client and Server Sessions.

Added Flexibility for Password and User ID Naming

SAS/CONNECT now supports the use of periods (.) and spaces in passwords and user-IDs for the PASSWORD= and USERNAME= options. The PASSWORD= option can be specified in the SIGNON statement or the RSUBMIT statement when signing on to a server session or submitting code to a remote SAS session.

Enhanced Password Support on z/OS

SAS/CONNECT now supports passwords that have mixed case on z/OS, and it supports the IBM standard for password phrases that have a length of up to 100 characters. For information about the IBM standard for password phrases, see Assigning Password Phrases in *z/OS Security Server RACF Security Administrator's Guide*.

Support for New Base SAS Language Elements

SAS/CONNECT supports the new SAS automatic macro variable SYSPROCESSMODE, which returns the name of the run mode or server type for the current SAS session. For more information about using SYSPROCESSMODE with SAS/CONNECT, see Using SYSPROCESSMODE to Display the Run Mode or Server Type.

Support for Extended Attributes

The UPLOAD and DOWNLOAD procedures in SAS/CONNECT now support the transfer of data containing extended attributes. Extended attributes are created and managed by specifying options in the MODIFY statement of PROC DATASETS. The new XATTR= option in SAS/CONNECT specifies whether to allow for the transfer of

extended attributes that are defined on a SAS data set or a SAS library. For more information, see XATTR=YES | NO.

For general information about extended attributes in Base SAS, see “Extended Attributes” in *SAS Language Reference: Concepts*. For syntax information about extended attributes in Base SAS, see the “DATASETS Procedure” in *Base SAS Procedures Guide*.

Support for the New Default Values of the LRECL= Option

Starting in SAS 9.4, the default logical record length (LRECL) that SAS allows for reading and writing external files has increased to 32767. If you are using fixed length records (RECFM=F), the default value for LRECL is 256. For more information about using the LRECL= option in SAS/CONNECT, see “Tips for Using PROC DOWNLOAD and PROC UPLOAD ” on page 94.

New Locked-Down State Restrictions

The LOCKDOWN statement and LOCKDOWN system option are new in Base SAS 9.4M1. With LOCKDOWN, the SAS server administrator can create a restricted environment in which there is limited access to specified directories and files. All other directories and files are inaccessible. In addition to there being access restrictions on directories and files, there are also restrictions on how you sign on when running a SAS session in a locked-down state. For more information, see “Locked-Down SAS Sessions” on page 26. Locked-Down SAS Sessions.

Enhanced INFILE= Option in PROC UPLOAD and PROC DOWNLOAD

In SAS/CONNECT 9.4M2 enhancements have been made to the INFILE option in PROC UPLOAD and PROC DOWNLOAD. Now, when selecting multiple files for upload or download using the INFILE option, you can use the wildcard character to specify 0 or more characters anywhere in the filename. For example, you can specify Report2*.txt to select all files beginning with Report2 and ending with .txt. In previous releases, the wildcard character could not be used to represent characters within a filename. This new pattern-matching capability enables you to more

efficiently transfer data to and from remote sessions. For more information, see INFILE=client-file-identifier.

Document Enhancements

In SAS/CONNECT 9.4M3 the *SAS/CONNECT User's Guide* was improved. Content from the *Communication Access Methods for SAS/CONNECT and SAS/SHARE* relevant to SAS/CONNECT software was moved to *SAS/CONNECT User's Guide* to provide easier access to all information related to SAS/CONNECT software.

Some of these changes include the following:

- a new administrative section that contains information about TCP/IP connections, signing on, and setting up the SAS/CONNECT spawner
- a new section describing SAS/CONNECT files and the terminology used to discuss them
- a new section defining the contents and scope of the document
- a reorganization of the SAS/CONNECT language elements into one comprehensive dictionary, entitled *Part 3: SAS/CONNECT Language Reference*
- a new section containing sign-on examples

New Authinfo File Support for Credentials

In SAS/CONNECT 9.4M5 support was added so that the user can supply credentials in an authinfo file instead of on a SIGNON statement. Use of an authinfo file is required if you want to connect from SAS 9.4M5 to SAS running in SAS Viya and connect to SAS Cloud Analytic Services (CAS).

New TCPPROXYLIST Environment Variable

In SAS/CONNECT 9.4M5 the TCPPROXYLIST environment variable were added to support HTTP_CONNECT so that SAS clients outside of the cloud can sign-on to SAS/CONNECT spawners. By setting the TCPPROXYLIST environment variable, you can connect to different clouds from the same client.

Changed Default Value for the TCPLISTENTIME Option

In SAS/CONNECT 9.4M5 the default value for the TCPLISTENTIME option was changed to 300. Previously, the default value was 0, or no time limit. The TCPLISTENTIME option is the amount of time a SAS/CONNECT server will listen for a SAS/CONNECT client to connect

New NOCLEARTEXT Default Spawner Behavior

In SAS/CONNECT 9.4M5 you no longer need to add the NOCLEARTEXT spawner option to increase security. The NOCLEARTEXT spawner option has been made the default value and is no longer valid as an option. The CLEARTEXT option has been added. to be used only when absolutely necessary because credentials are transmitted unencoded.

New Error Message for Sign-ons from Workspace Servers That Allow Numeric Session-ids

In SAS/CONNECT 9.4M5 users that sign on from workspace servers that allow numeric session-ids will now get an error message. The documentation has been updated to indicate that a server name must be 8 characters or less and start with an alphabetic character.

Added _USER_ Option to %SYSRPUT Statement

In SAS/CONNECT 9.4M5 the _USER_ option was added to the %SYSRPUT statement to enable all user-defined macro variables to be sent from the server to the client.

Deprecation of Telnet and CLEARTEXT option

With the June 2023 hot fix, Telnet is deprecated and it is recommended to use SAS/CONNECT Spawner for client sign-ons. The -CLEARTEXT option has been deprecated and is no longer available. For more information, see SAS Note 70114.

PART 1

Introduction to SAS/CONNECT

1

Introduction

About This Book

Overview

This document provides the following information:

- *Part 1: Introduction* – provides an overview of SAS/CONNECT software and the services that it offers.
- *Part 2: Using SAS/CONNECT* – contains conceptual and practical information about how to use SAS/CONNECT to perform various types of sign-ons and how to use the three services that are offered with SAS/CONNECT: compute services (CS), remote library services (RLS), and data transfer services (DTS).
- *Part 3: SAS/CONNECT Language Reference* – contains a dictionary of SAS/CONNECT language elements and their syntax.
- *Part 4: Administration* – contains information about the access methods that are used with SAS/CONNECT and the connection types that are available. This

section also contains information about how to manually set up and use the SAS/CONNECT spawner in a SAS Foundation environment.

- *Part 5: Logging and Debugging* – contains information about logging and troubleshooting sign-ons, TCP/IP connections, and data transfers.

This document is for SAS/CONNECT with SAS 9.4. For information about SAS/CONNECT with SAS Viya, see *SAS/CONNECT for SAS Viya: User's Guide*.

Document Scope

Administrative Sections

The sections contained in *Part 4: Administration* focus primarily on administrative tasks that are performed with SAS Foundation installations as opposed to those performed for planned deployments in which much of the initial configuration is done for you by the SAS Deployment Wizard. As such, this document describes the manual setup of the SAS/CONNECT spawner, as well as the access methods and connection types that are available with SAS/CONNECT software in a SAS Foundation environment.

Information for managing servers (including the SAS/CONNECT server) in a SAS Intelligence Platform deployment can be found in the SAS® Intelligence Platform Documentation.

For a list of documents related to setting up and managing the SAS/CONNECT spawner in a SAS Intelligence Platform environment, see “SAS/CONNECT in a SAS Intelligence Platform Environment” on page 5.

If you used the SAS Deployment Wizard to initially configure SAS/CONNECT in a planned deployment, but you want to make specific updates to the SAS/CONNECT spawner configuration without the use of the SAS Deployment Manager, you can manually configure the SAS/CONNECT spawner using the steps outlined in this document.

Usage and Language Reference Sections

Part 2: Using SAS/CONNECT and *Part 3: SAS/CONNECT Language Reference* contain usage and syntax information related to SAS/CONNECT software and are not specific to any particular SAS environment or SAS product. This information is intended to serve as the primary reference for SAS/CONNECT language elements in all SAS environments (including SAS Intelligence Platform deployments, SAS Foundation installations, and other scenarios that include SAS/CONNECT software).

SAS/CONNECT in a SAS Intelligence Platform Environment

The following resources contain information for managing the SAS/CONNECT spawner and server in a SAS Intelligence Platform environment:

Conceptual Information:

- Understanding SAS/CONNECT Servers in *SAS Intelligence Platform: Application Server Administration Guide*.
- The Uses of SAS/CONNECT in the SAS Intelligence Platform in *SAS Intelligence Platform: Application Server Administration Guide*

Server Administration, Installation, and Configuration:

- SAS 9.4 Intelligence Platform: Administration Documentation
- Initial Configuration of the SAS/CONNECT Server in *SAS Intelligence Platform: Application Server Administration Guide*
- Using Scripts to Operate SAS Servers Individually in *SAS Intelligence Platform: Application Server Administration Guide*.
- Using SAS Management Console to Operate the SAS Object Spawner or the SAS/CONNECT Spawner in *SAS Intelligence Platform: Application Server Administration Guide*.
- SAS 9.4 Intelligence Platform: Installation, Configuration and Migration Documentation

What is SAS/CONNECT?

Overview

SAS/CONNECT software is a SAS client/server toolset that provides the ability to manage, access, and process data in a distributed and parallel SAS environment. As a client/server application, SAS/CONNECT links a SAS client session to a SAS server session. The terms client and server depict the relationship between two SAS sessions. The client session is the initial SAS session that creates and manages one or more server sessions. The server session can run either on the same computer as the client (for example, on an SMP computer) or on separate hardware, such as on a remote computer across a network.

Features

SAS/CONNECT enables users and applications developers to achieve the following:

maintain SAS interoperability across architectures and SAS releases

- transfer disk copies of data
- directly process a remote data source and get results back locally
- develop local graphical user interfaces that process remote data sources

develop scalable SAS solutions

- run multiple independent processes asynchronously
- scale up to fully use the capabilities of symmetric multiprocessing (SMP) hardware
- scale out to fully use the features of distributed processors
- use pipeline processing (TCP/IP ports) to run multiple dependent processes asynchronously
- combine the resources of multiple computers to work in parallel

manage distributed resources

- perform daily or nightly automated backups
- initiate transaction processing to a master database at a specified time each day
- centralize and automate data and report distribution to workstations in a network
- centralize and automate data collection from workstations in a network

Note: *Asynchronous* Compute Services is commonly referred to as MP (Multi-Process) CONNECT.

Services

The SAS/CONNECT toolset offers 3 types of services:

- Compute Services
- Remote Library Services

- Data Transfer Services

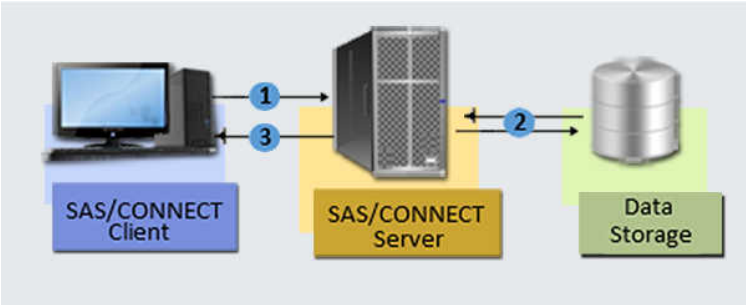
Compute Services

Compute Services That Use RSUBMIT

Compute Services provides access to all of the computing resources on your network by enabling you to direct the execution of SAS programs to one or more server sessions. An RSUBMIT block, or a “remote submit,” is a block of statements that are created in the client session using the RSUBMIT and ENDRSUBMIT statements. RSUBMIT blocks are executed in the remote server session. The results and any output that is generated by the remote execution are returned to the client session.

For short-running tasks, remote submits can be processed synchronously. This means that control is returned to the client session after the remote processing is complete. For longer-running tasks, remote submits can be processed asynchronously. This means that control is returned immediately, and you can continue local processing or remote processing to another server session.

Figure 1.1 *Model of Compute Services*



The figure shows that these services enable you to move some or all portions of an application's processing to a remote computer.

Compute Services enables you to do the following:

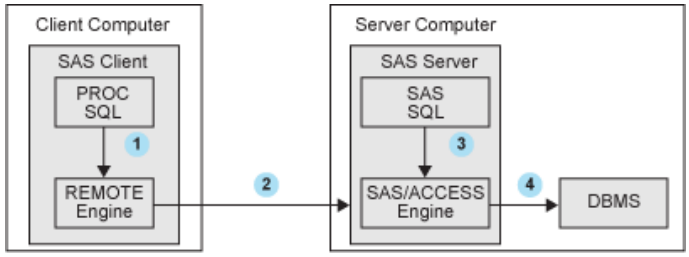
- achieve scalability for your SAS applications
  - perform remote tasks in the background (asynchronously) while processing locally
  - run multiple SAS processes asynchronously and coordinate the results from each task execution in your client SAS session
  - use pipeline processing to overlap execution of multiple dependent SAS DATA steps or procedures
  - use processors on an SMP computer (which is referred to as "scaling up") and using idle processors across a network (which is referred to as "scaling out")
- access remote resources

- take advantage of server hardware and software resources
- access mainframe and other legacy systems (for example, by building a single SAS program that contains statements that run locally and statements that execute on multiple remote legacy computers)
- execute against the remote copy of the data
- submit macro steps remotely to the server, and then pass return code information about the server process to the client
- execute graphics programs on the server and display the graphics locally by using the graphics capabilities of the local workstation, plotter, or printer

Compute Services That Use Remote SQL Pass-Through

Remote SQL pass-through (RSPT) gives you control of where SQL processing occurs. RSPT enables you to pass SQL statements to a remote SAS SQL processor by passing them through a remote SAS server. You can also use RSPT to pass SQL statements to a remote DBMS by passing them through a remote SAS server and a Remote access engine that supports pass-through.

Figure 1.2 *Remote SQL Pass-Through Services*



1. The SAS client uses a Remote engine to pass SQL statements to a server session.
2. The SQL statements are passed to the server session.
3. The SQL statements are passed to SAS SQL to select data or to execute statements in order to modify, manipulate, and manage data. This includes creating SAS SQL views.
4. The SQL statements are passed to a remote DBMS to select data or to execute statements in order to modify, manipulate, and manage data. This includes creating DBMS views.

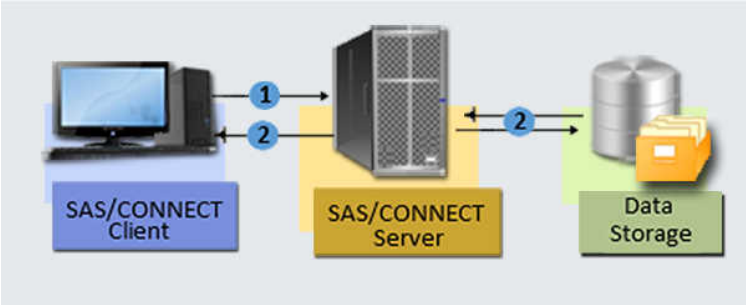
You can invoke RSPT by using PROC SQL statements that are passed to the remote server for execution in the server SAS session, or you can store SQL pass-through statements in local SQL views. For information about statements that are used for remote SQL pass-through, see “RSPT” on page 153.

For more information about using compute services, see “Overview of Compute Services” on page 30

Remote Library Services

Remote Library Services (RLS) provides transparent access to SAS data that is located on a remote computer. The data resides in server libraries, and RLS moves the data through the network as client processing requests it. The data must again pass through the network on any subsequent use by the client session. As the following figure shows, a copy of the data is not written to the client file system.

Figure 1.3 *Model of RLS Processing*



The SAS procedures and DATA steps that run in the SAS/CONNECT client session request access via the Remote engine to SAS files that are located on a SAS/CONNECT server. The Remote engine communicates the requests for data to the server. The server administers the requests to access SAS files on behalf of the client.

RLS provides the following:

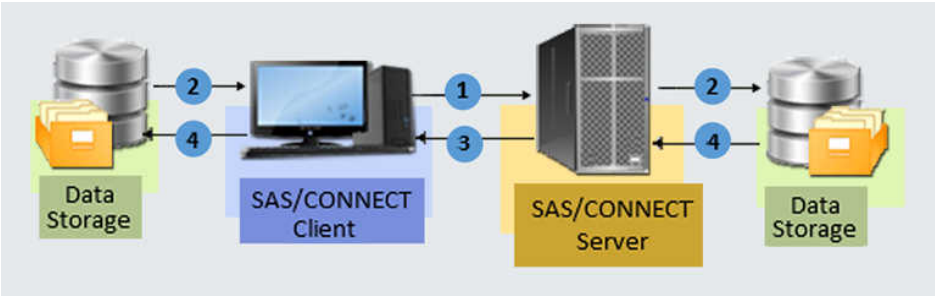
- transparent access to SAS data that is located on a remote computer
- access to current SAS data because no client copy is made
- a reduction of disk space consumption because multiple copies of the data are not created
- the ability to run a local graphical user interface and process SAS data that is located on a remote computer

For more information about remote library services, see Using Remote Library Services on page 72.

Data Transfer Services

Data Transfer Services enables you to move a copy of your data from one computer to another computer. The data is translated between computer architectures and SAS version formats, as necessary.

Figure 1.4 Model of Data Transfer Services (UPLOAD and DOWNLOAD)



Data is transferred using the UPLOAD and DOWNLOAD procedures. You can transfer SAS data sets, SAS catalogs, MDDB, SQL views, entire SAS libraries, and external text or binary files.

The data transfer capabilities enable you to do the following:

- customize data transfers
  - transfer multiple SAS files in a single step by using the INLIB= and OUTLIB= options. This capability enables you to transfer an entire library or selected members of a library in a single PROC UPLOAD or PROC DOWNLOAD step.
  - transfer collections of files (such as a partitioned data set, a MACLIB, or a directory) between a client and a server.
  - use WHERE processing for dynamic data subsetting and SAS data set options when transferring individual SAS data sets.
  - transfer catalog entries that contain graphics output by using a simple one-step process.
- protect data
  - increase the robustness of your decision support environment by keeping a local copy of your data, which is insulated from network failure.
  - back up local files to a server.
- manage data distribution
  - automate both data or application distribution and centralized data collection.
  - distribute files from one workstation by uploading to a server and downloading to other workstations that need the files.
  - move SAS files between releases of SAS as well as across operating environments.

For more information about using data transfer services, see Using Data Transfer Services on page 87.

Access Methods

TCP/IP is the supported access method on UNIX, z/OS, and Windows operating environments. The XMS access method is also supported when both client and

server sessions are on z/OS. For more information about TCP/IP Access Method, see “TCP/IP Access Method” on page 308.

Encryption Providers

Encryption providers include the SAS products and third-party strategies for protecting data and credentials (user IDs and passwords) that are exchanged in a SAS/CONNECT client/server environment. All these providers use proven, industry-standard encryption algorithms for data protection.

Here are the encryption providers that SAS/CONNECT can use:

SAS Proprietary
: is a fixed encoding algorithm that is included with Base SAS software. It requires no additional SAS product licenses. The SAS proprietary algorithm is strong enough to protect your data from casual viewing. SAS Proprietary provides a medium level of security.

SAS/SECURE
: is a product within the SAS System. In SAS 9.4, SAS/SECURE is included with Base SAS software. In prior releases, SAS/SECURE was an add-on product that was licensed separately. This change makes strong encryption available in all deployments (except where prohibited by import restrictions).

Transport Layer Security/Secure Sockets Layer (TLS/SSL)
: cryptographic protocols that are designed to provide communications security over a computer network. In addition to providing encryption services, TLS/SSL performs client and server authentication, and it uses message authentication codes to ensure data integrity.

Secure Shell (SSH)
: is a protocol that enables users to access a remote computer via a secure connection. SSH is available through various commercial products and as freeware. OpenSSH is a free version of the SSH protocol suite of network connectivity tools. Although SAS software does not include a programming interface to SSH functionality, SAS does support the tunneling feature of SSH that enables a SAS client to make an encrypted connection to a SAS server. Port forwarding is another term for tunneling. The SSH client and SSH server act as agents between the SAS client and the SAS server, tunneling information via the SAS client's port to the SAS server's port.

For details about these encryption providers, see *Encryption in SAS*.

PART 2

Using SAS/CONNECT

2

Signing On

Types of Sign-ons

Overview

There are several types of SAS/CONNECT sign-ons. These sign-on types can be grouped into two main categories: those that use SAS metadata and those that do not use SAS metadata.

- Non-metadata server-based sign-ons:
  - “Spawner Sign-ons” on page 16
  - “SASCMD (MP Connect) Sign-ons” on page 16
  - “Telnet Sign-ons” on page 17
- Metadata server-based sign-ons (in a SAS Intelligence Platform Deployment):
  - Sign-ons to a Logical SAS/CONNECT server on page 19

Sign-ons in a SAS grid server on page 19

Non-Metadata Server-based Sign-ons

Spawner Sign-ons

Spawner sign-ons occur when a SAS/CONNECT client uses TCP/IP to contact a SAS/CONNECT spawner running on a remote host to start a SAS session on that remote host. Here is an example of a sign-on to a UNIX server that is running the SAS/CONNECT spawner:

```
%let session1=xyz.mydomain.com 2324;
signon session1;
```

In the example, the name of the remote server on which the SAS session runs is **xyz.mydomain.com**. The spawner is listening for client requests on port **2324**. If no port is specified, the default port 23 is used. The session for this connection is named ‘session1.’

Using the SAS/CONNECT spawner to sign on eliminates the need for a sign-on script and it provides default encryption for a user ID and password. Signing on to a SAS/CONNECT spawner is preferred over signing on using a Telnet daemon because the SAS/CONNECT spawner provides a medium level of security through SAS Proprietary Encryption.

You can use an authinfo file to sign on to the spawner without having to specify credentials in a SIGNON statement. An authinfo file contains a user ID and password that is used for authentication. Use of an authinfo file is not available under the z/OS operating environment.

SAS/CONNECT first checks for credentials in the USER= and PASSWORD= options or the AUTHDOMAIN= option in the SIGNON statement. If you use an authinfo file, you must explicitly specify _AUTHINFO_ as the value in the PASSWORD= option in the SIGNON statement. If there is more than one user ID in your authinfo file that could be used to connect to the spawner, you should specify a value for the USER= option to select which one to use. If you specify_AUTHINFO_ and SAS/CONNECT fails to retrieve functional credentials from an authinfo file, the system generates an error message and the connection attempt fails.

For more information about the authinfo file, including how to create and format a file, see “Client Authentication Using an Authinfo File” in *Client Authentication Using an Authinfo File*.

SASCMD (MP Connect) Sign-ons

SASCMD signons can be established when you want to run multiple, independent SAS sessions on the same multiprocessor machine. Here is an example of a SASCMD sign-on:

```
signon session1 sascmd="!sascmd -nosyntaxcheck -noterminal";
```

```
    rsubmit session1 wait=no;
        <statements>;
    endrsubmit;

signon session2 sascmd="!sascmd -nosyntaxcheck -noterminal";
    rsubmit session2 wait=no;
        <statements>;
    endrsubmit;
signoff session1;
signoff session2;
```

Note: The global SASCMD option is defined by the system administrator as a restricted option by defining it in the rsasv9.cfg file. When the SASCMD= option is in a SIGNON statement, it fails and results in a warning that says : **`SASCMD option is restricted by the Site Administrator and cannot be changed. The restricted option value will be used.`** For more details, see Configuration Guide for SAS 9.4 Foundation for UNIX Environments - Restricted Options.

Telnet Sign-ons

IMPORTANT With the June 2023 hot fix, Telnet is deprecated and it is recommended that you use SAS/CONNECT Spawner for client sign-ons. The -CLEARTEXT option has been deprecated and is no longer available. For more information, see SAS Note 70114.

Telnet sign-ons use the Telnet program to connect to a remote server across a TCP/IP network without the use of the SAS/CONNECT spawner. In Telnet sign-ons, the SIGNON statement starts a Telnet session that connects to the remote host where a sign-on script is executed to start SAS. You must specify a sign-on script when signing on using Telnet.

Here is an example of a Telnet sign-on in which the FILENAME statement specifies the sign-on script and the OPTIONS statement specifies the name of the remote host:

```
filename rlink '!sasroot/misc/connect/tcptso.scr';
options remote=xyz.mydomain.com;
signon;
```

Metadata Server-based Sign-ons

SAS Metadata Server

If you are running SAS/CONNECT in a SAS Intelligence Platform environment, then there are other types of sign-ons that require you to access the SAS Metadata Server.

The SAS Metadata Server is a multi-user server that serves metadata from metadata repositories to all of the SAS Intelligence Platform client applications in your environment. The SAS Metadata Server holds the information for a collection of servers that can all be managed under one logical SAS Application Server. The SAS Application Server can contain multiple logical servers, including a SAS/CONNECT server and a SAS Grid server, that help make up the compute tier of your SAS architecture. In this context, SAS/CONNECT server properties and sign-on properties are stored as metadata in the metadata repository.

By accessing the metadata server, you can continue to execute SAS/CONNECT applications in the traditional interactive and batch execution modes, but with the convenient access to configured sign-on properties. This access means that you do not need to specify SAS options for signing on in your code. Once you establish a connection with the metadata server, you can use the SIGNON statement to sign on to the SAS/CONNECT server component of the SAS Application Server.

For both SAS/CONNECT server sign-ons and SAS Grid Server sign-ons, your client computer must be able to access the SAS Metadata Server. The SAS Metadata Server contains the definitions for the SAS/CONNECT server and SAS Grid Server in the SAS Metadata Repository. You can access the SAS Metadata Server by specifying certain SAS system options for metadata. Here is an example:

```
options metaserver="max.apex.na.com"
metaport=8561
metaprotocol="bridge"
metauser="domain\joe"
metapass="*******";
```

In this example, you submit the appropriate credentials to access the SAS Metadata Server, which runs on the computer `max.apex.na.com`. The bridge network protocol is used to communicate with the SAS Metadata Server via port 8561.

Note: If the client session is not configured to access the SAS Metadata Server, SAS displays a pop-up window in which you can configure access to the SAS Metadata Server.

Logical SAS/CONNECT Server Sign-ons

After you access the SAS Metadata Server, you can use the SIGNON statement to sign on to the SAS/CONNECT server component of the SAS Application Server. In the SAS Open Metadata Architecture, the metadata for a SAS Application Server specifies one or more server components that provide SAS services to a client. You must know the name of the SAS Application Server. Before sign-on, you can see a list of the configured sign-on properties for the SAS Application Server by specifying either the SERVER= or SERVERV= options in the SIGNON statement. In the following examples, the name of the SAS Application Server is SASApp:

```
signon server="SASApp";
```

or

```
signon serverv="SASApp";
```

Here is an excerpt of the output that is generated when the SIGNON serverv="SASApp" statement is executed:

```
1    options metaserver="max.apex.na.com";
2    signon serverv="SASApp";
NOTE: Server=                SASApp - Connect Server
      Remote Session ID=     remhost
      ServerComponentID=     A5SXFC1R.AU000002
      Remote Host=           max.apex.na.com
      Communication Protocol=TCP
      Port=                  7551
      AuthDomain= DefaultAuth
      Wait= Yes
      SignonWait= Yes
      Status= Yes
      Notify= No
```

After you view the sign-on properties, you can sign on to the server session. Here is an example:

```
signon server="SASApp";
```

A sign-on to the SAS Application Server that is named `SASApp` implies a SAS/CONNECT server sign-on.

Note: The output generated by the SERVERV= option includes properties that control server sign-on and server session execution. These connection properties are saved and stored in the metadata repository via SAS Management Console.

SAS Grid Server Sign-ons

Servers operating in a SAS grid environment are simply SAS/CONNECT servers that have been started out “on the grid” and that have been defined in a particular way on the metadata server. The primary difference between a SAS/CONNECT server that is grid-enabled and one that is not is how they are defined in metadata.

To sign on to a grid enabled server, you must have SAS Grid Manager installed and your client computer must be able to access the SAS Metadata Server. To access the SAS Metadata Server, specify the SAS system options for metadata.

After you have specified the metadata server options, specify the GRDSVC_ENABLE function followed by the SIGNON statement in your SAS/CONNECT client session. Specify _ALL_ as the value for the GRDSVC_ENABLE function to enable grid execution on all SAS server sessions. This enables grid execution on all subsequent sign-ons and remote submits to all server sessions.

If you use the GRDSVC_ENABLE function to enable grid execution on a specific server session, then only grid execution is enabled for all future sign-ons and remote submits to that server session.

When the user is connecting to SAS Workload Orchestrator using credentials that consist of a user ID and password, the credentials will either be retrieved from metadata or from an AUTHINFO file. If the metadata user has credentials stored in metadata for the logical grid server's authentication domain, those credentials will be used. Else, the AUTHINFO file for the current SAS user will be checked for credentials that match the host and port of the grid master.

In the following example, Section 1 establishes access to the metadata server. Section 2 uses the GRDSVC_ENABLE function with _ALL_ to enable grid execution for all server sessions. Section 3 disables grid execution for a specific server session, which, in this case, does not support grid execution.

```
1 options metaserver="max.apex.na.com"
      metaport=8561
      metaprotocol="bridge"
      metauser="domain\joe"
      metapass="*******";

2 %put gs_rc=%sysfunc(grdsvc_enable(_all_,server=SASApp));
      signon grid1;

3 %put gc_zos_rc=%sysfunc(grdsvc_enable(zos,""));
      %let zos=zoshost.mydomain.com 3456;
      signon zos;
      signoff _all_;
```

1 access the metadata server

2 enable grid execution for all server sessions

3 disable grid execution for a specific session (session with server-ID ZOS)

Note: To disable grid execution on one or all server sessions, specify the "" (empty string) option in the GRDSVC_ENABLE function.

For more information about the GRDSVC_ENABLE function, see GRDSVC_ENABLE in *Grid Computing in SAS*.

Where to Find More Information

This document provides more detailed information about signing on in a non-metadata server-based environment. This information is organized according to operating environment and can be found in the following locations:

- Chapter 20, “UNIX Operating Environment,” on page 339
- Chapter 21, “z/OS Operating Environment,” on page 361
- Chapter 22, “Windows Operating Environment,” on page 383

For more information about metadata server-based environments, see the SAS 9.4 Intelligence Platform: Administration Documentation web page at http://support.sas.com/documentation/onlinedoc/intellplatform/tabs/admin94.html.

Interfaces for Using SAS/CONNECT

Types of Interfaces for Using SAS/CONNECT

You can use SAS/CONNECT and start server sessions in any of these interfaces:

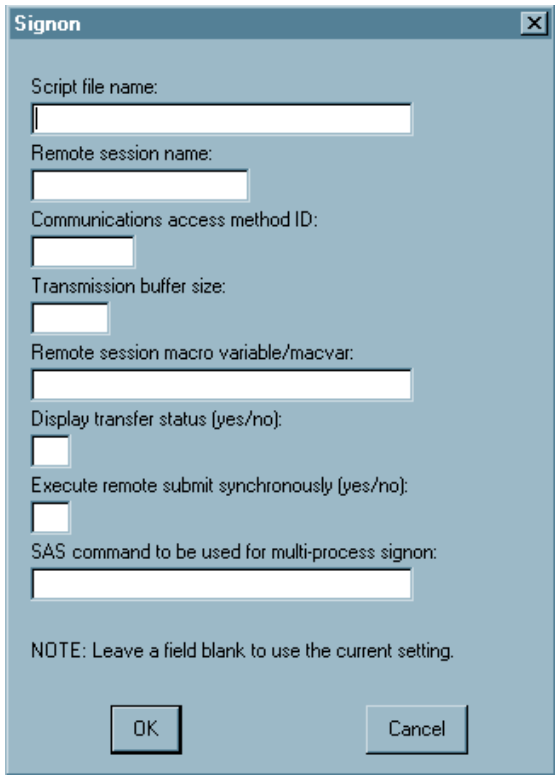
- SAS Windowing Environment
- Program Editor Window
- Autoexec File

The SAS Windowing Environment with SAS/CONNECT

The Sign-on Window

To start a SAS/CONNECT session:

1. Select **Run** ⇨ **Signon** from the menu bar in the SAS Program Editor window.
2. Complete the following fields in the Sign-on window.



Script file name:
If you use the TCP/IP access method and choose to use a script file, enter the full path and the name of the script file. For example, to connect to the z/OS operating environment by using the TCP/IP access method, enter the following: *pathname*`/tcptso.scr`

The default location of the script file varies according to operating environment.

Remote session name:
Enter the name of the session that you are connecting to. For details, see "CONNECTREMOTE=" on page 108.

Communications access method ID:
Enter the value for the COMAMID= option. For example, for the TCP/IP access method, enter the following: `tcp`

For complete details about access methods, see "Access Methods Supported by SAS/CONNECT" on page 307.

Transmission buffer size:
Enter the value of the buffer size that SAS/CONNECT uses for transferring data. For details, see "TBUFSIZE=" on page 121.

Remote session macro variable/macvar:
Enter the name of the macro variable that you want to use to associate with the server session. For details about the CMACVAR= option, see " CMACVAR=*value*" on page 148.

Display transfer status (yes/no):
Type `yes` or `no` to specify whether the status window is displayed during data transfers. For details, see "CONNECTSTATUS" on page 110.

Execute remote submit synchronously (yes/no):
Type `yes` or `no` to specify whether remote submits are to be executed synchronously or asynchronously.

YES
specifies synchronous remote submits, which means that control is not returned to the client session until the remote submit is finished processing. This is the default.

NO
specifies asynchronous remote submits, which means that control is immediately returned to the client session after processing begins on the server session.

For details, see "CONNECTWAIT" on page 111.

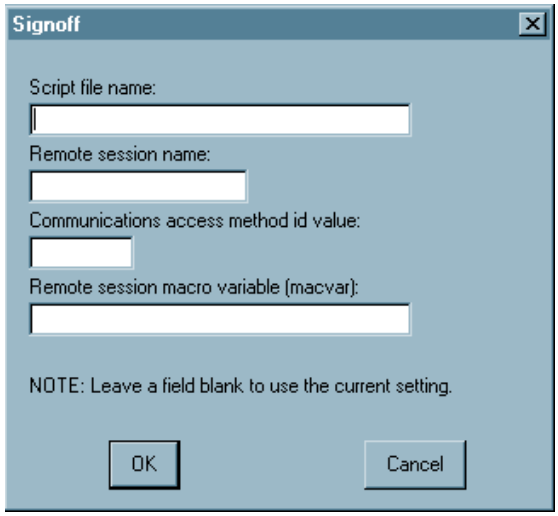
SAS command to be used for multi-process signon:
If you do not use SMP hardware, omit this field. If you use SMP hardware, specify a command and options in this field to invoke a server session that executes on the multiprocessor computer. For details about multiprocessing, see "MP CONNECT" on page 31.

Note: If you have defined an RLINK fileref, you must clear the reference as follows: `filename rlink clear;`

3 Select **OK** to sign on, or select **Cancel** to return to the Program Editor window without signing on.

The Sign-off Window

1 To stop a SAS/CONNECT session by signing off, from the menu in the Program Editor window, select **Run** ⇨ **Signoff**.



2 If you are signed on to only one server session, you can click **OK** to end that session.

If you are signed on to multiple server sessions, verify that the field entries are valid for the session that you want to end.

The Program Editor Window with SAS/CONNECT

Use the Program Editor Window to Sign On SAS/CONNECT

1 Enter an OPTIONS statement in the Program Editor window of the client session.

Use the SUBMIT command, statement, or function key to execute the OPTIONS statement. You use the OPTIONS statement to specify the COMAMID= and REMOTE= system options. For example:

```
options comamid=communications-method
   remote=server-ID;
```

Note: The REMOTE= option is an alias for the CONNECTREMOTE= system option.

For details about specifying values for these options, see “COMAMID=” on page 101 and “CONNECTREMOTE=” on page 108.

2 Issue the SIGNON command or enter the SIGNON statement in the client session. Specify the appropriate sample script (if necessary) for the operating environment:

```
signon cscript='external-file-name-of-script';
```

Note: Sample automatic sign-on scripts should be modified with installation-specific information before you can use them to start the connection.

Here is an example of signing on to a server that is running a spawner program:

```
remote=nodename servicename;
signon user=_prompt_;
```

After the SIGNON command executes successfully, a message in the Log window indicates that the connection is established.

Use the Program Editor Window to Sign Off SAS/CONNECT

Issue the SIGNOFF command, or enter the SIGNOFF statement in the client session:

```
signoff;
```

After the SIGNOFF command executes successfully, a message in the Log window indicates that the connection has ended.

Note: If you used a script to sign on, the same script can be used to stop the connection. The sample scripts that are used for automatic sign-on are used for signing off your server session.

Use the Autoexec File with SAS/CONNECT

The autoexec file contains SAS statements that can be executed automatically when you begin a client session. You can simplify the process of starting and stopping the connection by following these recommendations:

- Include a FILENAME statement in the autoexec file that defines the fileref RLINK. Make sure that it gives the correct file specification for the script that you use to start SAS/CONNECT. For details, see “FILENAME” on page 201.

  By assigning the fileref RLINK to your script, you can start the connection without specifying the script name in the SIGNON command.

  Also, you can stop the connection without specifying the script name in the SIGNOFF command because RLINK is the reserved fileref for script files.

  When SAS executes a SIGNON or a SIGNOFF command without a fileref, SAS automatically searches for a file that is defined with RLINK as the fileref. If RLINK has been defined, SAS executes the corresponding script.

- Include an OPTIONS statement in your autoexec file to specify the COMAMID= and CONNECTREMOTE= system options.

  Windows Example:

  ```
  options comamid=tcp
     remote=remhost;
  ```

  Using the autoexec file to specify system options is a convenience over having to execute an OPTIONS statement in each SAS session when using SAS/CONNECT.

Modifying your autoexec file as recommended eliminates a step in the process of starting the connection, and you can use the short form of the SIGNON and SIGNOFF commands.

For example, to start a connection from a SAS session that was invoked by using a modified autoexec file, issue the SIGNON command or submit the SIGNON statement:

```
signon
```

or

```
signon;
```

After you have completed your server processing, in order to end the connection, issue the SIGNOFF command or submit the SIGNOFF statement :

```
signoff
```

or

```
signoff;
```

For more information about the autoexec file, see the information for your operating environment:

- "Customizing Your SAS Session By Using Configuration and Autoexec Files" in *SAS Companion for UNIX Environments*
- "Autoexec Files" in *SAS Companion for z/OS*
- "Uses for the Autoexec File" in *SAS Companion for Windows*

Locked-Down SAS Sessions

Sign On to Locked-Down SAS Sessions

Overview

The LOCKDOWN feature allows administrators to limit access to files for SAS processes that are executing in batch or server processing mode. When a SAS process is running in a locked-down state, these resources are accessible to the user:

- resources that are specified in the *lockdown path list*[1]
- libref and filerefs that are defined in the autoexec file
- pre-assigned libraries that are defined in metadata
- source code repositories for defined stored process programs

When LOCKDOWN is in effect, there is limited access to files, and there are restrictions on how you can sign on.

For more information about locked-down SAS sessions, see SAS Intelligence Platform: Security Administration Guide.

SASCMD Sign-ons

If the SAS process that you are running is in a locked-down state, then you can create SAS/CONNECT server sessions on your local machine under the following conditions:

- Only "!SASCMD" and "!SASCMDV" are valid as values for the SASCMD= option. If you specify a script file or command as the value for the SASCMD= option, the sign-on will fail. For example, the following error message is displayed when an invalid value is specified for the SASCMD= option:

1. The lockdown path list is typically created and maintained by the system administrator to make specified files available and not subject to the lockdown.

```
ERROR: Only "!SASCMD" or "!SASCMDV" are allowed for the SASCMD option
when the LOCKDOWN option is specified. ERROR: Remote signon canceled.
```

- Any additional options that are specified after the SASCMD option is specified are ignored. For example, the sign-on in the following example will be successful, but the TBUFSIZE option will be ignored, and there will be a WARNING in the log.

```
signon sess1 sascmd="!sascmdv -tbufsize 1024";
WARNING: Additional options after !SASCMD are ignored when the
LOCKDOWN option is specified.
```

Scripted Sign-ons

If you are doing a scripted sign-on, the script file must be available to the client session. If SAS is in a locked-down state on the local machine and the script file is not in a path accessible to the client, then the sign-on will fail. In the following example, a client attempts to use the **tcpwin.scr** file to perform a scripted spawner sign-on. The script file is not defined in the lockdown path list, so an invalid path error is displayed in the log and the sign-on fails:

```
filename rlink "C:\Program Files\SASHome\SASFoundation\9.4\connect\
     saslink\tcpwin.scr";
ERROR: The path
C:\Program Files\SASHome\SASFoundation\9.4\connect\saslink\
tcpwin.scr is not in the list of
accessible paths when SAS is in the lockdown state.
ERROR: Error in the FILENAME statement.
```

Server Security

The following steps should be taken when locking down a SAS/CONNECT server:

- Specify the LOCKDOWN option in the SAS configuration file, and define a lockdown path list in the SAS autoexec file.
- Start the SAS/CONNECT spawner using the **-NOSCRIPT** option. This prevents users from gaining access to the system by inserting operating system commands into the script file.
  - When starting the spawner with the -NOSCRIPT option, either specify the spawner's SASCMD command in the spawner start-up, or define the SASCMD command in the SAS/CONNECT server's metadata. The spawner uses this command to start the SAS/CONNECT server session.
- Ensure that the SAS/CONNECT spawner is not started with the **-SHELL** option. As long as the -SHELL option is *not* specified, the **-NOXCMD** option will be added by default to the server's invocation parameters. **-NOXCMD** prevents clients from executing X commands from their SAS sessions to access system files.

Logging

If a user attempts to access a resource that is locked down, SAS issues an error message to the SAS log. If the SAS session is configured for the SAS Logging Facility, SAS issues an error message to the Audit.Lockdown logger. Log files that are defined in the logging facility configuration file are not limited by lockdown and can be used for debugging purposes.

3

Using Compute Services

Overview of Compute Services

SAS/CONNECT Compute Services provides a set of statements and commands that enable the client to distribute SAS processing to one or more server sessions and to maintain control of these server sessions and their results from the single client session. This very powerful capability enables you to run SAS across many (possibly heterogeneous) platforms as well as communicate between different releases of SAS that might be installed on these operating environments.

The RSUBMIT statement or command is used to direct SAS processing to a specific server session. For details, see “RSUBMIT” on page 161.

Here are some of the benefits of Compute Services:

- gives you access to additional CPU resources.

  You might have multiprocessor SMP computers or remote computers on your network that are underused. These CPUs could be used to execute the CPU intensive portions of your application faster and more efficiently than your local computer. Compute Services enables you to move some or all segments of an application to one or more server sessions for execution and return the results to the client session.

- lets you execute the application on the computer where the data resides.

  Data center rules or data characteristics might mandate a single, centralized copy of the data that is needed by your application. Moving the processing to the computer where the data resides eliminates the need to transfer or create additional copies of the data. Using only one copy of data can satisfy security requirements as well as enable access to data sources that are too large or too dynamic for transfer.

  For example, although data links between computers make file transfers convenient and easy, large files do not move quickly between computers. It is also inefficient to maintain multiple copies of large files when developing and testing programs that are designed to process those files. Compute Services overcomes this limitation by developing applications on one computer while running them and keeping the data that they use on a different computer.

  To test your application, submit it remotely from the client session so that it will run in the server session on a remote computer. All processing occurs on the computer where the data resides, but the output appears in the client session.

MP CONNECT

MP CONNECT

Prior to SAS 8, when an RSUBMIT statement was executed, the client session was suspended until processing by the server session had completed. In SAS 8, MP CONNECT functionality was added, which enables you to execute RSUBMIT statements asynchronously. When an RSUBMIT is executed asynchronously, the unit of work is sent to the server session and control is immediately returned to the client session. The client session can continue with its own processing or execute RSUBMIT statements to one or more additional server sessions. Asynchronous RSUBMIT statements are most useful for longer-running tasks.

MP CONNECT enables you to perform multiprocessing with SAS by establishing a connection between multiple SAS sessions and enabling each of the sessions to asynchronously execute tasks in parallel. You can also merge the results of the asynchronous tasks into your local execution stream at the appropriate time. In addition, establishing connections to processes on the same local computer has been greatly simplified. This enables you to exploit SMP hardware as well as network resources to perform parallel processing and easily coordinate all the results into the client SAS session.

You can use MP CONNECT to start any number of SAS processes that you want to perform in parallel. SAS processes that are started on a single multiprocessor computer are independent, unique processes just as they are if they are initiated on a remote host. For example, under Windows and UNIX, each SAS session is a separate process that has its own unique SAS Work library. Each process also assumes the user context of the parent or of the user that invoked the original SAS session, and has all the rights and privileges that are associated with that parent. Under z/OS, each SAS session is an MVS BPX address space that inherits the same STEPLIB and USERID as the client address space. The client's SASHELP, SASMSG, SASAUTOS, and CONFIG allocations are passed to the new session as SAS option values.

MP CONNECT is implemented by executing an RSUBMIT statement and the CONNECTWAIT=NO option. This method causes SAS/CONNECT to submit a task to a server session for processing and return control immediately to the client session so that you can start other tasks in the client session or in other server sessions. For details about the CONNECTWAIT= option, see “RSUBMIT” on page 161.

Independent Parallelism

Overview

Independent parallelism is possible when the execution of Task A and Task B do not have any interdependencies. For example, an application might need to run PROC SORT against two different SAS data sets and merge the sorted data sets into one final data set. Because there is no dependency between the two data sets that initially need to be sorted, the two SORT procedures can be performed in parallel. When sorting is complete, the merge can take place. MP CONNECT can be used to accomplish independent parallelism.

MP CONNECT can also be used to start multiple SAS sessions to execute independent units of work in parallel. The client session can synchronize the execution of the parallel tasks for subsequent processing. For this example, two SAS sessions would be started, and each session would perform one of the SORT procedures. The merge would be executed in the client session after the two parallel SORT procedures are completed.

Considerations for Independent Parallelism

When using MP CONNECT (especially on an SMP computer), ensure that the implementation of parallel sessions does not create an I/O bottleneck in one or both of the following areas:

- single input data source
- I/O activity in the Work library of each SAS session

Single Input Data Source

If a single input data source is being read by each of the parallel SAS sessions, overall execution time can actually be longer if all the parallel SAS sessions are trying to read their input from a single disk and single I/O channel. One way to solve this bottleneck would be to create multiple copies of your data on separate disks or mount points. Another way would be to create subsets of your data on multiple mount points, and have each parallel session process a different subset of the data. In addition, you could enable multi-user access to a single large data source by using the new Scalable Performance Data Engine (SPD Engine), which is available in SAS 9. The SPD Engine accelerates the processing of large data sets by accessing data that has been partitioned into multiple physical files called partitions. The SPD Engine initiates multiple threads with each thread having a direct path to a partition of the data set. Each partition can then be accessed in parallel (by a separate processor), which allows the application to analyze data in parallel as fast

as the data is read from disk. This can effectively reduce I/O bottlenecks and substantially decrease the amount of time that is used to process data.

I/O Activity in the Work Library of Each SAS Session

The I/O activity in the Work library for a typical SAS process can be very high. When you use MP CONNECT to start multiple SAS sessions on the same SMP computer, each session has its own Work library. The Work libraries for these processes are all created in the same temporary directory by default. As a result, you might have multiple SAS processes performing intensive I/O in the same directory on the same physical disk, causing an I/O bottleneck. This problem can be minimized in one of two ways.

- Use the Work invocation option on each of the MP CONNECT processes to direct each process to create its Work library on a separate disk.
- Use the SPD Engine to create a temporary library to be used instead of the Work library, and point the USER= option to this temporary library. The SPD Engine can partition data sets over multiple file systems. Utility data sets that are created by SAS procedures continue to be stored in the Work library. However, any data sets that have one-level names and that are created by your SAS programs are stored in the User library.

Note: When using MP CONNECT on multiple remote computers, the Work library of the remote sessions exists on the individual computers, so this bottleneck does not occur.

Pipeline Parallelism

Overview of Pipeline Parallelism

Pipeline parallelism occurs when the execution of Task A and Task B have interdependencies. For example, a SAS DATA step might be followed by a PROC SORT of the data set that is created by the DATA step. PROC SORT is dependent on the execution of the DATA step, because the output of the DATA step is the input needed by PROC SORT. However, the execution of the two steps can be overlapped, and the DATA step can pipe its output into PROC SORT. The piping feature of MP CONNECT provides pipeline parallelism.

Piping enables you to overlap the execution of SAS DATA steps and some SAS procedures. This is accomplished by starting one SAS session to run one DATA step or SAS procedure and piping its output through a TCP/IP socket as input into another SAS session that is running another DATA step or SAS procedure. This pipeline can be extended to include multiple steps and can be extended between different physical computers. Piping improves performance not only because it

enables overlapped task execution, but also because intermediate I/O is directed to a TCP/IP pipe instead of written to disk by one task and then read from disk by the next task.

Piping is implemented by using a LIBNAME statement to identify a port to be used for the pipe. For details about using the LIBNAME statement to implement piping, see “LIBNAME: SASESOCK Engine” on page 209. For an example of piping, see “Example 6: MP CONNECT with Piping” on page 64.

Limitation of Pipeline Parallelism

A limitation of piping is that it supports single-pass, sequential data processing. Because piping stores data for reading and writing in TCP/IP ports instead of disks, the data is never permanently stored. Instead, after the data is read from a port, the data is removed entirely from that port and the data cannot be read again. If your data requires multiple passes for processing, piping cannot be used.

Here are some examples of SAS procedures and statements that process single-pass, sequential data:

- DATA step
- SORT procedure
- SUMMARY procedure
- GANTT procedure
- PRINT procedure
- COPY procedure
- CONTENTS procedure

Considerations for Piping

- The benefit of piping should be weighed against the cost of potential CPU or I/O bottlenecks. If execution time for a SAS procedure or statement is relatively short, piping is probably counterproductive.
- Ensure that each SAS procedure or statement is reading from and writing to the appropriate port.

  For example, a single SAS procedure cannot have multiple writes to the same pipe simultaneously or multiple reads from the same pipe simultaneously. You might minimize port access collisions on the same computer by reserving a range of ports in the SERVICES file. To completely eliminate the potential for port collisions, request a dynamically allocated port instead of selecting an explicit port for use. For details, see “LIBNAME” on page 205.
- Ensure that the port that the output is written to is on the same computer that the asynchronous process is running on. However, a SAS procedure that is reading from that port can be running on another computer.
- Ensure that the task that reads the data does not complete before the task that writes the data. For example, if one process uses a DATA step that is writing observations to a pipe and PROC PRINT is running in another task that is

reading observations from the pipe, PROC PRINT must not complete before the DATA step is complete. This problem might occur if the DATA step is producing a large number of observations, but PROC PRINT is printing only the first few observations that are specified by the OBS= option. This would result in the reading task closing the pipe after the first few observations had been printed, which would cause an error for the DATA step, which would continue to try to write to the pipe that had been closed.

Note: Although the task that is writing generates an error and will not complete, the task that is reading will complete successfully. You could ignore the error in the writing task if the completion of this task is not required (as is the case with the DATA step and PROC PRINT example in this item).

- Be aware of the timing of each task's use of the pipe. If the task that is reading from the pipe opens the pipe to read and there is a delay before the task that is writing actually begins to write to the pipe, the reading task might time-out and close the pipe prematurely. This could happen if the writing task has other steps to execute before the DATA step or SAS procedure that is actually writing to the pipe.

  Use the TIMEOUT= option in the LIBNAME statement to increase the time-out value for the task that is reading. Increasing the value for the TIMEOUT= option causes the reading task to wait longer for the writing task to begin writing to the pipe. This will allow the initial steps in the writing task to complete and the DATA step or SAS procedure to begin writing to the pipe before the reading task time-out expires. For an example, see “Example 7: Prevent Pipes from Closing Prematurely” on page 65.

Benefits of MP CONNECT

MP CONNECT can greatly reduce the total elapsed time that is required to execute your SAS applications that contain tasks that can be executed in parallel. MP CONNECT provides a syntactic interface to distribute multiple units of work across idle CPUs either on the same SMP computer or across multiple computers on your network.

MP CONNECT uses hardware resources that you might have thought were outdated and useless. Using MP CONNECT, you can put multiple, slow, inexpensive computers to work in parallel on a job, transforming them into a powerful and inexpensive computing resource.

Large jobs that previously never finished executing can be implemented via MP CONNECT to repeatedly distribute small pieces of a problem to multiple processors until the entire problem is solved.

MP CONNECT enables you to use SAS in cluster and grid environments for high-performance computing.

Piping enables you to overlap the execution of one or more SAS DATA steps and procedures in order to accelerate processing. Piping has the added benefit of eliminating the need to write intermediate SAS data sets to disk, which not only saves time but reduces the physical disk space requirements for your SAS processing.

Scalability with MP CONNECT

Overview of Scalability

Scalability reduces the time-to-solution for your critical tasks. Scalability can be accomplished by performing two or more tasks in parallel (independent parallelism) or overlapping two or more tasks (pipeline parallelism). Scalability requires two things: 1) that some part(s) of your application can be overlapped or performed in parallel, and 2) that you have hardware that is capable of multiprocessing. All applications are not scalable, and not all hardware configurations are capable of providing scalability.

To decide whether an application can be scaled, consider the following questions:

- Does the time that is required to run a job exceed the batch window of time that you have available?
- Does the time that is required to run a job allow enough time so that you can make appropriate decisions after you get the information from the application? The applications that are the best candidates for scalability generally take hours, days, or maybe even weeks to execute.
- Can the application (or some part of it) be segmented into sub-tasks that are independent and can be run in parallel? It might be worthwhile to duplicate some data in order to achieve this independence.
- Does the application contain dependent steps that could benefit from piping?

Hardware that is capable of multiprocessing includes an SMP computer or multiple computers on a network with each computer containing one or more processors. In addition to the number of processors, it is important to have multiple I/O channels. This is inherent to multiple computers on a network. For an SMP computer, this can be accomplished with RAID arrays that enable you to stripe or spread your data across multiple physical disks. Even for a single threaded application, this can improve I/O performance, because the operating system is able to read data from multiple drives simultaneously and synchronize the result for the application.

Parallel Threads and Parallel Processes

SAS 9 has the capability to leverage the available hardware resources to both scale up and scale out your applications. SAS provides scalability in two ways:

- parallel SAS processes
- parallel threads within a SAS process

Parallel Processes

A SAS process consists of many pieces, including execution units, data structures, and resources. A process corresponds to an operating environment process. A process has a largely private address space. It is scheduled by the operating environment, and its resources are managed by the operating environment at the lowest level. Multiple SAS processes use multiple processors on an SMP computer, but they can also be run on multiple remote single or multiprocessor computers on a network. When running multiple SAS processes on an SMP computer, SAS does not schedule a specific process to a specific processor; scheduling is controlled by the operating environment. MP CONNECT provides the ability to run multiple SAS processes.

Parallel Threads

A process consists of one or more threads. A thread is also scheduled by the operating environment, but the running process might influence the behavior of threads by using synchronization techniques. All threads in a process share an address space and must cooperatively share the resources of the process. Multiple threads use multiple processors on an SMP computer but cannot be executed across computers. When running multiple threads within a SAS process, SAS does not schedule a specific thread to a specific processor; scheduling is controlled by the operating environment.

Scaling Up

Scaling up means to increase the number of processors, disk drives, and I/O channels on a single server computer. Scaling up also means to leverage the multiple processors, disk drives, and I/O channels on a single server computer.

Scaling Out

Scaling out means adding more hardware, not bigger hardware. Scaling out also means to exploit network resources to run parts of an application. When you scale out, the size and speed of an individual computer does not limit the total capacity of the network.

Multiple Threads and Multiple Processors

Beginning in SAS 9, multiple threads are used to scale up and use multiple processors in SMP hardware. Multithreading has been incorporated into SAS 9 (and later), including many SAS servers, several performance-critical SAS procedures,

and many SAS engines. Multithreading is used for both computing-intensive parts as well as I/O-intensive parts in order to process data quickly and reduce the total execution time.

Multiple SAS processes (MP CONNECT) are used to both scale up and scale out. By running multiple processes on an SMP computer, the operating environment can schedule the processes on different processors to use all the hardware resources on the computer. In addition, by running multiple SAS processes across the computers that are available on a network, you can use idle processors and put multiple, slow, inexpensive computers to work in parallel on a job and turn them into a valuable, powerful, inexpensive computing resource.

Multithreading and multiple SAS processes (MP CONNECT) are not mutually exclusive. For some applications, the greatest gains in performance result from applying a solution that incorporates multiple threads and multiple processes. Provided you have the hardware resources to support it, you can use MP CONNECT to run multiple SAS processes and each process can use multithreading. When running multiple processes by using multiple threads on an SMP computer, it might be necessary to set SAS system options in each of the SAS processes to tune the amount of threading that is performed by each process. Tuning threading behavior avoids the sum of the processes and threads from overloading your system. When using multiple remote computers with each SAS process running on a physically separate computer, it might be better to let the threading within the process fully use the individual computers.

Successfully scaled performance is not obtained by installing more and faster processors or more and faster I/O devices. Scalability involves making choices about investing in SMP hardware, upgrading I/O configurations, using networked computers, reorganizing your data, and modifying your application. True scalability results from choosing scalable hardware and the appropriate software that is specifically designed to leverage it. The extent of the original problem that can be processed in parallel determines the amount of scalability that is achievable from the software solution.

Monitor MP CONNECT Tasks

Overview of Monitoring MP CONNECT Tasks

To monitor MP CONNECT tasks, the RDISPLAY command or statement creates two windows that enable you to view the contents of the accumulated server log and output without interrupting the asynchronous processing of the remote submitted task. The two windows enable you to view the accumulated log and output before merging them into your client session's log and output windows. For details about the syntax for the RDISPLAY command or statement, see “RDISPLAY” on page 182 and “RDISPLAY” on page 214.

As an alternative to RDISPLAY, you can use the SAS Explorer Monitor. For details, see “Use SAS Explorer to Monitor SAS/CONNECT Tasks” on page 40.

Manage MP CONNECT Log and Output Results

The log and output results that are generated by MP CONNECT server sessions are sent back to the client session as they are created. Because MP CONNECT tasks and client session tasks are processing in parallel, by default, the log and output are spooled to a utility file for later retrieval. If the log and output lines were written to the client Log and Output windows as they were produced, the output from MP CONNECT tasks and client session tasks would be interleaved, and the interpretation of the results of the executions would be impossible.

The MP CONNECT task log and output results can be viewed in separate windows using the RDISPLAY command or statement. For details, see “RDISPLAY” on page 182 and “RDISPLAY” on page 214.

Log and output results can also be written to, retrieved from, or merged in the client session Log and Output windows by using the RGET statement or command or redirecting to a file by using the LOG= option and the OUTPUT= option. For details about RGET, see “RGET” on page 183. For details about the LOG= option and the OUTPUT= option, see “RSUBMIT” on page 161.

MP CONNECT Task Completion

You can use any of the following to test for the completion of MP CONNECT tasks:

- LISTTASK statement
- SAS/CONNECT Monitor window from the SAS Explorer window
- CMACVAR macro variable
- NOTIFY=YES option
- WAITFOR statement

The LISTTASK statement lists information about a single active task by name or about all tasks in the current session. For details, see “LISTTASK” on page 197.

The SAS Explorer provides a menu selection that enables you to monitor SAS/CONNECT tasks that are executing asynchronously (or synchronously) in one or more server sessions. For details, see “Use SAS Explorer to Monitor SAS/CONNECT Tasks” on page 40.

The CMACVAR macro variable can be programmatically queried to learn the processing status (completed, failed, in progress) of an MP CONNECT task. For details, see “RSUBMIT” on page 161.

The NOTIFY=YES option requests the display of a notification message window to report the completion of an MP CONNECT task. For details, see “RSUBMIT” on page 161.

The WAITFOR statement makes the current SAS session wait for the completion of one or more asynchronously executing tasks that are already in progress. For details, see “WAITFOR” on page 195.

Use SAS Explorer to Monitor SAS/CONNECT Tasks

SAS Explorer provides a menu selection that enables you to monitor SAS/CONNECT tasks that are executing in one or more server sessions. A server session can execute across a network, or it can execute on a computer that is equipped with SMP hardware, which facilitates multi-processing.

To start the SAS/CONNECT Monitor, from the menu, select: **View** ⇨ **SAS/CONNECT Monitor**.

The SAS/CONNECT Monitor displays information about the tasks in two columns: Name and Status.

```
Name                Status

Task1               Complete
Task2               Running Asynchronously
Task3               Running Synchronously
```

The list of tasks is dynamically updated as new tasks start, and the **Status** field changes from **Running** to **Complete**, as appropriate. When you use the SIGNOFF statement to end a connection, the task is automatically removed from the window.

Note: If you do not see both columns, select **View** ⇨ **Details**.

You can also end a task that is running asynchronously by clicking the task in the Monitor and selecting the **Kill** option from the menu that is displayed when you right-click the mouse button. Similarly, you can select the **RDisplay** option from the menu to display a Log and Output window for a task that is running asynchronously.

Compute Services and the Output Delivery System

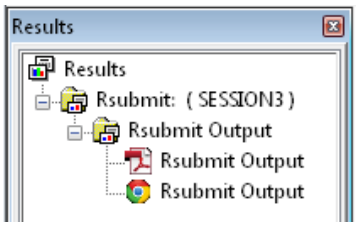
You can use the SAS Output Delivery System (ODS) to format the SAS output that is generated in a SAS session that runs on a server either synchronously or asynchronously.

Here are four typical programming scenarios for using Compute Services with ODS to manage output that is produced in a server session.

- Remotely submit procedure statements without any ODS statements.

  Any output that is produced by the remote submit produces a node in the Results window that has the name `Rsubmit:(`*`server-ID`*`)`. The Results window

uses ODS to generate pointers (nodes) to various positions in the Output window. The resulting node is a record of the output that is generated during a SAS server session.



- Precede and end the remote submit block (RSUBMIT through ENDRSUBMIT) with the appropriate ODS opening statement (such as ODS HTML or ODS PDF) and the corresponding ODS closing statement (such as HTML CLOSE or PDF CLOSE). Appropriate results are produced in the SAS session at the client. For example, ODS HTML produces output in the Results Viewer. ODS PDF produces output in the Results window.

```
ODS pdf;
rsubmit;
      <statements>;
endrsubmit;
ods pdf close;
```

- Precede RSUBMIT with the ODS OUTPUT statement.

  The output from the RSUBMIT appears in the Results window and is saved as a SAS data set.

```
ODS pdf;
rsubmit;
      <statements>;
endrsubmit;
```

- Remotely submit ODS statements and procedures and DATA step statements to produce the ODS output in the server session.

  The output is processed and generated entirely in the server session. Therefore, the results (for example, a SAS data set or HTML output) must be downloaded from the server session to the client session.

```
rsubmit;
       ODS pdf;
       <statements>;
endrsubmit;
```

For all scenarios that use asynchronous processing, use the “RGET” on page 183. The output is not available until the results are retrieved. The accumulated output is retrieved and transferred to the client session.

For details about ODS, see the *SAS Output Delivery System: User’s Guide*.

Use the SAS Windowing Environment to Control Remote Processing

Overview of Remote Processing Control Using the SAS Windowing Environment

The SAS windowing environment includes menu selections that enable you to control remote processing during a SAS session. The following Compute Services menu selections are available from the `Run` menu:

Remote Submit
: enables you to submit one or more statements to a SAS/CONNECT server session for remote processing.

Remote Get
: merges the spooled Log and Output lines from the asynchronous remote submit operation with the client's Log and Output windows for viewing.

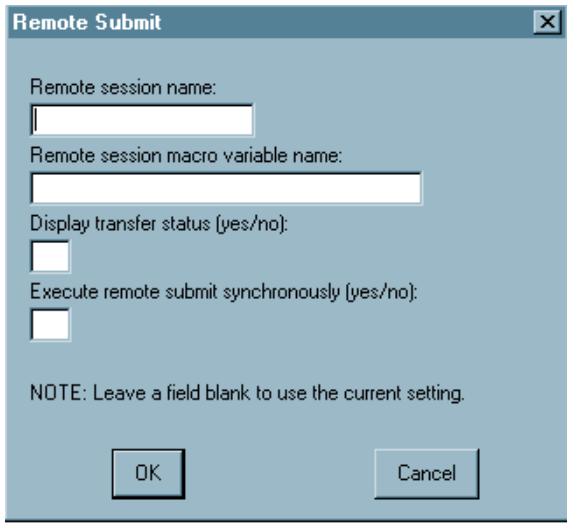
Remote Display
: enables you to view the spooled Log and Output lines that are created by the asynchronous remote submit operation in the Log and Output windows that are created for the specific remote server session.

Remote Submit

To submit one or more statements to a SAS/CONNECT server session for remote processing, open the SAS Program Editor window and select **Run** ⇨ **Remote Submit** from the menu bar.

The Remote Submit dialog box appears.

Figure 3.1 Remote Submit Dialog Box



Here are explanations of the fields:

Remote session name
specifies the server session that the statements are executed in. If only one session is active, this field can be empty. If multiple server sessions are active, omitting the remote session name causes the program statements to be run in the session that is specified in the CONNECTREMOTE= option. You can find out which server session is current by examining the value that is specified in the CONNECTREMOTE system option.

For information about the CONNECTREMOTE= option, see "RSUBMIT" on page 161.

Remote session macro variable name
associates a macro variable with a specific RSUBMIT block. Macro variables are especially useful for controlling the execution of multiple asynchronous RSUBMIT operations.

For information about the CMACVAR= option, see "RSUBMIT" on page 161.

Display transfer status (yes/no)
specifies whether the status window for file transfers is displayed for the current remote submit operation.

If this field is empty, the default value is obtained from the CONNECTSTATUS= system option or the CONNECTSTATUS= option in the SIGNON= statement for this server.

For information about the CONNECTSTATUS= option, see "RSUBMIT" on page 161.

Execute remote submit synchronously (yes/no):
specifies whether the remote submit operation executes synchronously or asynchronously. Synchronous processing means that server processing must be completed before control is returned to the client session. Asynchronous processing permits the client and one or more server session processes to execute in parallel. Control is returned to the client session immediately after a remote submit begins execution to allow continued processing in the client session.

If the field is empty, the default value is obtained from the CONNECTWAIT= system option or the CONNECTWAIT= option in the SIGNON= statement for this server.

For information about the CONNECTWAIT= option, see “RSUBMIT” on page 161.

Remote Submit Limitation:

CAUTION

The Remote Submit menu cannot be used if a CARDS statement, a CARDS4 statement, a DATALINES statement, a DATALINES4 statement, or a PARMCARDS statement is included in the remote submit operation. The Remote Submit menu is prohibited from processing data because of its implementation as a macro. A macro definition cannot contain a CARDS statement, a DATALINES statement, a PARMCARDS statement, or data lines.

However, you can use any of the following methods to execute a remote submit that contains any of these statements:

- Enter the RSUBMIT command in the command window.
- Enter the RSUBMIT and ENDRSUBMIT statements in the editor window.
- Submit the statements for local execution, and then use PROC UPLOAD to transfer the created output to the server session.

Remote Get

To merge the spooled log and output from the asynchronous remote submit operation with the client's Log and Output windows for viewing, open the SAS Program Editor window and select **Run** ⇨ **Remote Get** from the menu bar.

Here are explanations of the fields:

Remote session name
specifies the server session whose spooled log and output lines are to be merged into the client's Log and Output windows. If only one session is active, this field can be empty. If multiple server sessions are active, omitting the remote session name causes RGET to execute for the session that is specified in the CONNECTREMOTE= option.

For more information, see “RGET” on page 183.

Note: **Remote Get** applies only to asynchronous remote submit operations. If you execute **Run** ⇨ **Remote Get** while the asynchronous remote submit operation is in progress, the operation is automatically converted to synchronous processing so that all of the lines from the server session can be merged.

Note: To view the spooled Log and Output lines that are created by the asynchronous remote submit operation (does not merge with the client's Log and Output windows), select **Remote Display**.

Remote Display

To view only the spooled Log and Output lines from the asynchronous remote submit operation, open the SAS Program Editor window and select **Run** ⇨ **Remote Display** from the menu bar.

Here are explanations of the fields:

Remote session name
specifies the session name of the server whose Log and Output lines are to be viewed. If only one session is active, this field can be empty. If multiple server sessions are active, omitting the remote session name causes RDISPLAY to execute in the session that is specified in the CONNECTREMOTE= option.

For more information, see “RDISPLAY” on page 182.

Note: **Remote Display** applies only to asynchronous remote submit operations.

Note: To merge the spooled Log and Output lines that are created by the asynchronous remote submit operation with the client's Log and Output windows, select **Remote Get**.

Use the Macro Facility with SAS/CONNECT

Overview

When using the RSUBMIT statement within a macro definition, it is important to understand what code is compiled and executed locally versus what code is submitted to the server for execution. Understanding this distinction will help you when using macros and SAS/CONNECT software together.

This section discusses

- how compiled code and text behave when they are submitted remotely within a macro
- options and functions that can help you with these types of macros
- techniques for creating macro variables on the local and remote hosts

See “Macro Processing” in *SAS Macro Language: Reference* for more information about the SAS Macro Facility.

Submit Code Remotely Using a Macro

In SAS/CONNECT, you can use RSUBMIT blocks to separate server-session statements from client-session statements. Statements inside the RSUBMIT block are executed in the server session and all other statements are executed in the local session. However, this behavior can change when you use a macro with an RSUBMIT statement to remotely submit code.

If you want to create a macro that will submit SAS code to a remote server, you can do this by embedding an RSUBMIT block within a macro definition. We sometimes refer to these types of macros as “macro-generated RSUBMITs.”

When a macro is compiled, two results are produced: compiled macro statements and text. Even though they exist within the RSUBMIT block, these compiled macro statements, or instructional code, is executed in the local SAS session. Only the macro-generated text is passed to the remote server where it is executed remotely.

Understanding this distinction between what is passed along as text and what is compiled and executed locally is important if you want to use macros with RSUBMIT blocks.

Here is a complete list of code elements in SAS that are interpreted by the macro facility as text and therefore executed remotely:

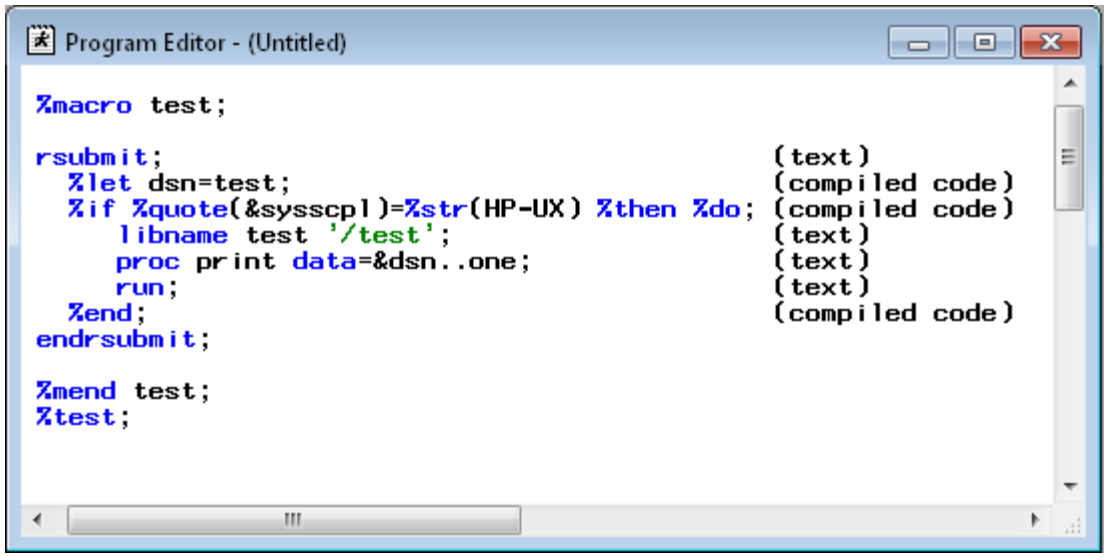
- macro variable references
- nested macro definitions and invocations
- macro functions, except %STR and %NRSTR
- arithmetic and logical macro expressions
- names and values of local macro variables
- text to be written by %PUT statements
- non-macro statements such as procedures and DATA step code
- field definitions in %WINDOW statements. This applies to SAS/CONNECT software since you cannot RSUBMIT a macro window.

Here are some items that are compiled by the macro facility and executed locally:

- %LET
- %IF
- %DO

In the example below, the statements in the macro definition are labeled according to how they are handled by the macro processor. Code that is compiled executes on the local machine and code that is read as text executes on the remote server.

Figure 3.2 How Macro-generated RSUBMIT Statements Are Interpreted by the Macro Processor



In the example below, if you were connecting from Windows to UNIX, the %IF statement condition would resolve to “false” because the statement would be compiled and processed in the local SAS session, which is running on Windows. Since the %IF statement resolves to “false,” then the statements following it are never executed, leaving nothing to submit to the remote host.

```
%macro test;
      rsubmit;
      %let dsn=test;
      %if %quote(&sysscpl)=%str(HP-UX) %then %do;
         libname test '/test';
         proc print data=&dsn..one;
         run;
      %end;
      endrsubmit;
   %mend test;
   %test;
```

To help you determine what parts of the macro statement are interpreted as text and what parts are considered compiled code, you can use the MLOGIC and MPRINT system options.

MPRINT and MLOGIC Macro System Options

The MLOGIC macro system option identifies and displays the instructional (compiled) code that is executed locally. The MLOGIC option specifies whether the macro processor prints a message whenever SAS executes any macro instructional code within a macro. Any statements produced by the MLOGIC option occur on the local host and everything else executes on the remote host.

The MPRINT macro system option identifies and displays the code that executes on the remote host. The MPRINT option displays SAS statements generated by macro execution. Any statements produced by the MPRINT option that appear between

the RSUBMIT ENDRSUBMIT block happen on the remote host and everything else executes on the local host.

The following example illustrates the MLOGIC and MPRINT macro system options:

Example Code 3.1 *Using the MPRINT and MLOGIC Macro System Options to Determine Where Your Code Is Executing*

```
options mlogic mprint;
   %macro test;
      rsubmit;
         data one;
            x=100;
         run;
         %let y=200;
         %put &y;
      endrsubmit;
   %mend;
   %test;
```

The following is written to the SAS log:

Example Code 3.1 *MPRINT and MLOGIC Log Output*

```
NOTE: Remote signon to HOST complete.
139
140 options mlogic mprint;
141 %macro test;
142 rsubmit;
143 data one;
144 x=100;
145 run;
146 %let y=200;
147 %put &y
148 endrsubmit;
149 %mend;
150 %test
MLOGIC(TEST): Beginning execution.
MPRINT(TEST): rsubmit
NOTE: Remote submit to HOST commencing.
MPRINT(TEST): ; data one;
MPRINT(TEST): x=100;
MPRINT(TEST): run;
MLOGIC(TEST): %LET (variable name is Y)
MLOGIC(TEST): %PUT &y
200
1 data one;
2 x=100;
3 run;
NOTE: The data set WORK.ONE has 1 observations and 1 variables.
NOTE: DATA statement used:
real time 0.23 seconds
cpu time 0.02 seconds
NOTE: Remote submit to HOST complete.
MPRINT(TEST): endrsubmit;
MLOGIC(TEST): Ending execution.
```

Notice that the MPRINT option shows the text that is pushed to the remote host; it consists of the DATA step. The MLOGIC option shows the compiled statements that remain on the local host. These are the %LET and %PUT statements.

See Also

- "MPRINT" in *SAS Macro Language: Reference*
- "MLOGIC" in *SAS Macro Language: Reference*

The %NRSTR Function

You can use the %NRSTR macro function to "hide" certain macro statements from the macro processor during compile-time. Hiding them prevents the macro processor from compiling and executing the specified statements locally. Instead, the function tells the SAS macro processor to interpret the statement as text and to pass it along to the remote session for execution. Here is an example of using the %NRSTR function:

```
%nrstr(%put abc=&abc one=&one time=&time;)
```

The following example illustrates what happens without the %NRSTR function:

Example Code 3.2 *Using a Macro-generated RSUBMIT without the %NRSTR Function*

```
%macro test;
      %put &sysscp;
      rsubmit;
         %let x=100;
         data new;
            put "&x";
         run;
         %put &sysscp;
      endrsubmit;
   %mend test;
   %test;
```

The following is written to the SAS log:

Example Code 3.2 *Output for a Macro-generated RSUBMIT without the %NRSTR Function*

```
MLOGIC(TEST): Beginning execution.
MLOGIC(TEST): %PUT &sysscp
WIN
MPRINT(TEST): rsubmit
NOTE: Remote submit to HOST commencing.
MLOGIC(TEST): %LET (variable name is X)
MPRINT(TEST): ; data new;
MPRINT(TEST): put "&x";
MPRINT(TEST): run;
MLOGIC(TEST): %PUT &sysscp
WIN
16 data new;
17 put "&x";
WARNING: Apparent symbolic reference X not resolved.
18 run;
&x
NOTE: The data set WORK.NEW has 1 observations and 0 variables.
NOTE: DATA statement used:
real time 0.02 seconds
cpu time 0.00 seconds
NOTE: Remote submit to HOST complete.
MPRINT(TEST): endrsubmit;
MLOGIC(TEST): Ending execution.
```

If this code was submitted on a Windows platform and a connection was established to an HP platform, the first %PUT would execute on the local host and print “WIN” in the SAS log. The RSUBMIT would run, but two of the items within the macro-generated RSUBMIT block, the %LET and %PUT statements, would be executed on the local host. The DATA step would be pushed to the REMOTE host and executed there. This would generate a warning because the %LET statement that defined the macro variable executed on the local host, rather than the remote host, where it is being called.

Here is the same example with the %NRSTR function added:

Example Code 3.3 *Using a Macro-generated RSUBMIT Used with the %NRSTR Function*

```
%macro test;
     %put &sysscp;
     rsubmit;
        %put &sysscp;
        %nrstr(%let x=100;)
        data new;
           put "&x";
        run;
        %nrstr(%put &sysscp;)
     endrsubmit;
  %mend test;

 %test;
```

The following is written to the SAS log:

Example Code 3.3 *Output for a Macro-generated RSUBMIT Used with the %NRSTR Function*

```
MLOGIC(TEST): Beginning execution.
MLOGIC(TEST): %PUT &sysscp
WIN
MPRINT(TEST): rsubmit
NOTE: Remote submit to HOST commencing.
MLOGIC(TEST): %PUT &sysscp
WIN
31 %let x=100;
32 ;
33 data new;
34 put "&x";
35 run;
100
NOTE: The data set WORK.NEW has 1 observations and 0 variables.
NOTE: DATA statement used:
real time 0.02 seconds
cpu time 0.01 seconds
36 %put &sysscp;
HP 800
NOTE: Remote submit to HOST complete.
MPRINT(TEST): ; %let x=100;
MPRINT(TEST): data new;
MPRINT(TEST): put "&x";
MPRINT(TEST): run;
MPRINT(TEST): %put &sysscp;
MPRINT(TEST): endrsubmit;
MLOGIC(TEST): Ending execution.
```

If this code was submitted on a Windows platform and a connection has been established to an HP platform, the first %PUT statement would execute on the local host and print “WIN” to the SAS log. The RSUBMIT statement would run but this time everything within the RSUBMIT would execute on the remote host, as shown by the MPRINT log output. When the DATA step executes on the remote host, the **x** variable resolves without a warning because the %NRSTR function allows the %LET statement to be executed on the remote host. The %NRSTR function also allows the %PUT statement to executed on the remote host.

See Also

“%NRSTR” in *SAS Macro Language: Reference*

The %SYSLPUT and %SYSRPUT Statements

Another issue that you might encounter when using SAS/CONNECT software and macros occurs when using macro variables. Many times, the macro variable is created on the local host and resolution tries to take place on the remote host or vice versa. The %SYSLPUT and %SYSRPUT statements can help with this issue.

The %SYSLPUT statement creates a new macro variable or modifies the value of an existing macro variable on a remote host or server. The syntax for %SYSLPUT varies across releases of SAS.

In the SAS 6 and 7 releases, %SYSLPUT is a SAS sample program with the following syntax:

```
%SYSLPUT(macro-variable,value,remote=);
```

In the SAS 8 release, %SYSLPUT is a macro statement with the following syntax:

```
%SYSLPUT macro-variable=value;
```

In the SAS 8 release, there is also a SAS sample program called %LPUT with the following syntax:

```
%LPUT(macro-variable,value,remote=);
```

In the SAS 9 release, %SYSLPUT is a macro statement that contains a new option with the following syntax:

```
%SYSLPUT macro-variable=value </remote=server-id>;
```

macro-variable is either the name of a macro variable or a macro expression that produces a macro variable name. The name can refer to a new or existing macro variable on a remote host or server.

value is a string or a macro expression that yields a string. Omitting the value produces a null (0 characters). Leading and trailing blanks are ignored. To make them significant, enclose the value in the %STR function.

To use the %SYSLPUT statement, you must establish a successful SIGNON between the local SAS session or client and a remote SAS session or server.

The following example shows how to use %SYSLPUT to create a macro variable called **Dir1** on the remote host:

Example Code 3.4 *Using %SYSLPUT to Create a Macro Variable on the REMOTE Host*

```
%macro test;
      %let dir1=/dept/test;
      %syslput dir1=&dir1;
      rsubmit;
         filename eng101 '/bin/sasfiles';
         proc upload infile= eng101 outfile="&dir1/eng101";
         run;
      endrsubmit;
   %mend test;
   %test;
```

In the SAS 9 release, a new option for the %SYSLPUT statement enables you to specify the name of the session in which the macro variable is created.

If only one session is active, the *server-id* can be omitted. If there are multiple server sessions active, omitting this option causes the macro to be created in the most recently accessed server session.

You can find out which server session is current by examining the value assigned to the CONNECTREMOTE system option.

The /REMOTE= option that is specified with the %SYSLPUT macro statement overrides the CONNECTREMOTE= global option.

Due to the addition of the /REMOTE option in the %SYSLPUT statement, any value that contains forward slashes should be quoted with a macro quoting function.

The following example uses the %BQUOTE function to mask forward slashes that are used in a UNIX path-name that is assigned in the %SYSLPUT statement:

Example Code 3.5 *Using the %BQUOTE Function with %syslput to Mask Forward Slashes in a UNIX Pathname*

```
%let path=/testa/testb;
  %syslput path=%bquote(&path);
  rsubmit;
     %put &path;
  endrsubmit;
```

The following is written to the SAS log:

Example Code 3.4 *Output for the %BQUOTE Function with %syslput*

```
NOTE: Remote submit to HOST complete.
917 %let path=/testa/testb;
918 %syslput path=%bquote(&path);
919 rsubmit;
NOTE: Remote submit to HOST commencing.
5 %put &path;
/testa/testb
NOTE: Remote submit to HOST complete.
```

The following example illustrates what occurs if the macro variable contains a forward slash and a macro quoting function is not used:

Example Code 3.6 *Using a Macro Variable That Contains a Forward Slash without a Macro Quoting Function*

```
%let path=/testa/testb;
   %syslput path =&path;
   rsubmit;
      %put &path;
   endrsubmit;
```

The following is written to the SAS log:

Example Code 3.5 *Output When Using a Macro Variable That Contains a Forward Slash without a Macro Quoting Function*

```
NOTE: Remote submit to HOST complete.
8 %let path=/testa/testb;
9 %syslput path=&path;
ERROR: Unrecognized option to the %SYSLPUT statement.
NOTE: Line generated by the macro variable "PATH".
1 /testa/testb
-
180
ERROR 180-322: Statement is not valid or it is used out of
proper order.
10 rsubmit;
NOTE: Remote submit to HOST commencing.
2 %put &path;
/testa/testb
NOTE: Remote submit to HOST complete.
```

The error is generated because once **&path** resolves, the first thing that is seen is the forward slash, so SAS assumes that the REMOTE= option is coming up next. Since the option is not there, an error occurs. This is not an issue in SAS releases prior to SAS 9, because the option did not exist.

The following table shows how to use the %SYSLPUT macro statement based on what version of SAS you are running.

Table 3.1 *The %SYSLPUT Statement*

Local Host	Remote Host	Usage
SAS 6 and 7 releases	SAS 8 release	use %SYSLPUT sample program
SAS 8 release	SAS 8 release and beyond	use %SYSLPUT macro statement
SAS 8 release	SAS 6 and 7 releases	use %SYSLPUT sample program

Note: %SYSLPUT sample program can be used when connecting from a SAS 8 release to a SAS 8 release if the REMOTE= option is needed.

To do the opposite of the %SYSLPUT statement, you use the %SYSRPUT macro statement. The %SYSRPUT statement assigns the value of a macro variable on a remote host to a macro variable on the local host. Here is the only syntax for %SYSRPUT:

```
%SYSRPUT local-macro-variable=value;
```

local-macro-variable specifies the name of a macro variable on the local host.

value is a macro variable reference or a character string on the remote host that is assigned to the *local-macro-variable*.

The following example uses the %SYSRPUT statement to assign a macro variable on a remote host to a macro variable on the local host:

Example Code 3.7 *Using the %SYSRPUT Statement to Assign a Remote Macro Variable to a Local Macro Variable*

```
rsubmit;
   %macro download;
      proc download data=remote.mydata out=local.mydata;
      run;
      %sysrput retcode=&sysinfo;
   %mend download;
   %download;
   endrsubmit;
   %macro checkit;
      %if &retcode = 0 %then %do;
        <further processing on local host>
      %end;
   %mend checkit;
   %checkit;
```

This section describes what happens when you place RSUBMIT blocks inside macro definitions. In many cases, you can move the RSUBMIT block outside the macro definition if you are getting error messages or unexpected results. By doing this, the macro itself is complied on the remote host and there is no question about where the code is executing. The MLOGIC and MPRINT options can also help you debug and determine what is being submitted remotely.

See Also

- "%SYSLPUT" in *SAS Macro Language: Reference*
- "%SYSRPUT" in *SAS Macro Language: Reference*

Use SYSPROCESSMODE to Display the Run Mode or Server Type

SYSPROCESSMODE is a Read-Only automatic macro variable that you can use to display the name of the SAS session run mode or server type. For example, you can use **&sysprocessmode** with a **%PUT** macro statement in the RSUBMIT block to have the server type, "SAS CONNECT Session," display in the log output, as shown in the following program:

```
SIGNON session1 sascmd="!sascmd -nosyntaxcheck -noterminal";
rsubmit;
    %put &sysprocessmode;
endrsubmit;
signoff session1;
```

Below is the partial log output for this program:

```
NOTE: Remote signon to SESSION1 complete.
      rsubmit;
NOTE: Remote submit to SESSION1 commencing.
       %put &sysprocessmode;
SAS Connect Session
```

For more information about SYSPROCESSMODE, see "SYSPROCESSMODE" in *SAS Macro Language: Reference*.

Compute Services and Break Windows

Overview

Break windows are a special class of windows for SAS/CONNECT client/server connections. Break windows enable you to handle error conditions that cause interruptions in processing by issuing a control-break signal. SAS provides two break windows to enable you to handle system interruptions and error conditions:

- Communication Services Break Handler window

- SAS/CONNECT attention handler window

These break windows also enable you to interrupt processing. Depending on which program statements are executing, you might see either of these break windows.

The Communication Services Break Handler window contains selections for actions that you can take in response to a problem or an interruption. Invoking the SAS/CONNECT attention handler window is one of the actions that you can select. Usually, you select the attention handler window to cancel statements that you have submitted to the server.

SAS/CONNECT Attention Handler Window

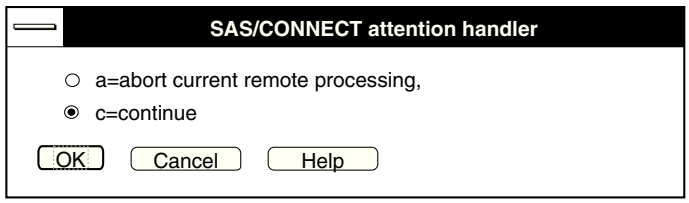
If you need to interrupt processing of statements that were submitted to the server, issue a break signal:

Table 3.2 *Break Signals*

Windows	Ctrl-Break
UNIX	Ctrl-C (This key combination can be reset with the UNIX STTY command. During a SAS session in DMS mode under the X Window System, you can select an interrupt button in the SAS Session Manager window to issue a break signal.) When you issue Ctrl-C, position the cursor in the window in which the SAS session was invoked.
z/OS	Attn key

After you issue a break signal, the SAS/CONNECT attention handler window appears as follows.

Figure 3.3 *The SAS/CONNECT Attention Handler Window*



The following selections are available in the attention handler window:

a
terminates the statements that are currently being processed in the server session but continues the connection to the server session. This option is useful if you want to terminate a very large file transfer, or if you want to interrupt a remote SAS job that is generating many error messages.

Note: Control might not be passed back to the client session immediately.

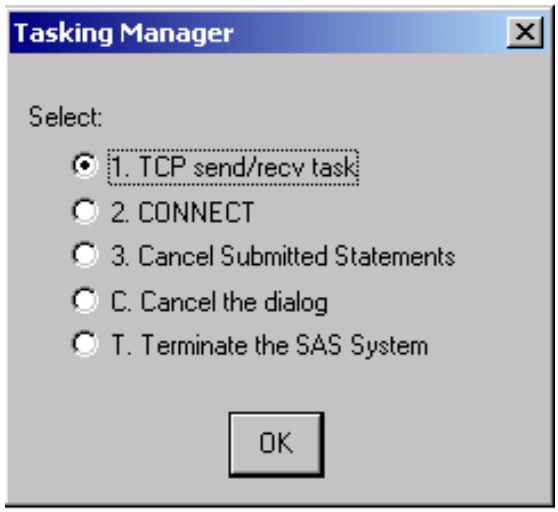
C
continues the remote job. Select this option if you decide that you do not want to interrupt the remote job.

Communication Services Break Handler Window

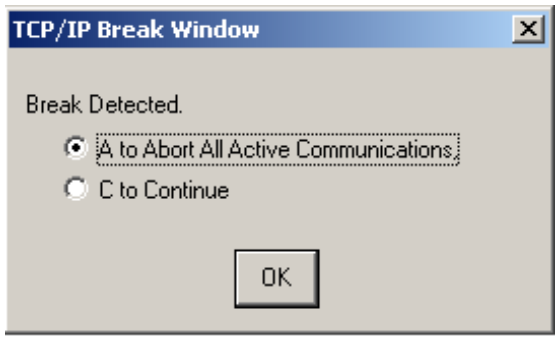
If the application detects an error condition, the Communication Services Break Handler window is displayed.

The following selections are available in the Communication Services Break Handler Window:

- `Ctrl-Break` displays the Tasking Manager window.



- Selecting **1. TCP send/recv task** displays the TCP/IP Break window.



- Selecting **2. CONNECT** displays the SAS/CONNECT attention handler window.

Examples Using Compute Services

Example 1: MP CONNECT for a Long-Running Remote Task

Purpose

This long-running program calculates summary statistics from the variables in a large SAS data set and downloads the summary statistics to your client session. The program also defines the macro variable REMSTATUS to store the status of the server task and uses the fileref REMLOG to store the log lines.

Program

```
rsubmit wait=no macvar=remstatus log=remlog;
libname remtdata 'external-file-name';
  proc summary data=remtdata.clinic;
     class diagnose;
     var age income visits;
     output out=sumstat
        n= mean= mage mincome mvisits;
   run;

   proc download data=sumstat out=summary;
   run;
endrsubmit;
```

Example 2: Administer Server Data Sets from a Client

Purpose

From a client session, you can use Compute Services to perform administration tasks on data sets that are located on the server.

This program administers password protection to the Tasklist data set and backs up a data set that is named Current.

Program

```
rsubmit;
   proc datasets lib=tsolib;

   modify tasklist (alter=sesame); 1
   run;

   age current backup1 - backup7; 2
   run;
   quit;
endrsubmit;
```

1 Add password SESAME to server data set Tasklist.

2 Maintain a week's worth of backup copies of data set Current.

Example 3: The CMACVAR= Option with MP CONNECT

Purpose

The following example enables you to remotely submit processing in a server session and allows the client session to immediately continue processing, and then retrieve and merge the results upon completion of that process.

The following program submits a PROC SORT and a PROC PRINT statement to be executed asynchronously in a server session. This server process is tested for completion by using the macro variable DONE.

Program

```
rsubmit cwait=no cmacvar=done;
   proc sort data=permdata.standard(keep=fname
      lname major tgpa gender)
      out=honor_graduates(where=(tgpa>3.5));
      by gender;
   run;

   title 'Male and Female Honor Graduates';
   proc print;
      by gender;
   run;
endrsubmit;

%macro get_results_when_complete;
   %if &done=0 %then %do;
     %put Remote submit complete,
        issuing "rget" to get the results.;
     rget;
   %end;
   %else %do;
     %put Remote submit not complete.;
     %put Issue:
        "%nrstr(%%)get_results_when_complete"
        later.;
   %end;
%mend;
%get_results_when_complete;

/* Continue with client session processing. */
/* Issue again if RSUBMIT isvnot complete.  */

%get_results_when_complete;
```

Example 4: The Output Delivery System with SAS/CONNECT

Purpose

ODS enables you to format and change the appearance of a procedure's output. The output is converted into objects that can be stored in HTML or in a SAS data set and can be manipulated and viewed in different ways.

This program creates, in a server session, a SAS data set and a SAS view that contain information about U.S. Presidents. The program then generates ODS

output. The first half of this example creates HTML from the SAS data set and SAS view. The second half uses ODS to create a SAS data set from the SAS view.

Program

```
signon rmthost sascmd="!sascmd -nosyntaxcheck -noterminal";
rsubmit;
   data presidnt; 1
      length fname lname $8 party $1 lady1 $10;
      input fname lname party year_in lady1;
      label fname='First Name'
            lname='Last Name'
            party='Party'
            year_in='Start Year'
            lady1='First Lady'
            ;
   datalines;
John Kennedy D 1961 Jackie
Lyndon Johnson D 1963 LadyBird
Richard Nixon R 1969 Pat
Gerald Ford R 1974 Betty
Jimmy Carter D 1977 Rosalynn
Ronald Reagan R 1981 Nancy
George Bush R 1989 Barbara
Bill Clinton D 1993 Hillary
George Bush R 2001 Laura
Barack Obama D 2009 Michelle
   ;
   run;

   proc sql nocheck; 2
      create view democrat as
      select fname,lname,party,lady1
         from presidnt
         where party='D';
   quit;

endrsubmit;

filename rsub '/u/myuserid/rsub.html' mod; 3
filename rsubc '/u/myuserid/rsubc.html' mod;
filename rsubf '/u/myuserid/rsubf.html' mod;
ods html
   file=rsub;
   contents=rsubc
   frame=rsubf
   ;

proc sql nocheck;
   connect to remote (server=rmthost);
title 'Democrats';
   select fname,lname,lady1
      from connection to remote
      (select * from democrat);
```

```
quit;

ods html close;

ods output output="rdata"; 4
title 'Republicans';
rsubmit;
   proc print data=presidnt;
   where party='R';
   run;
```

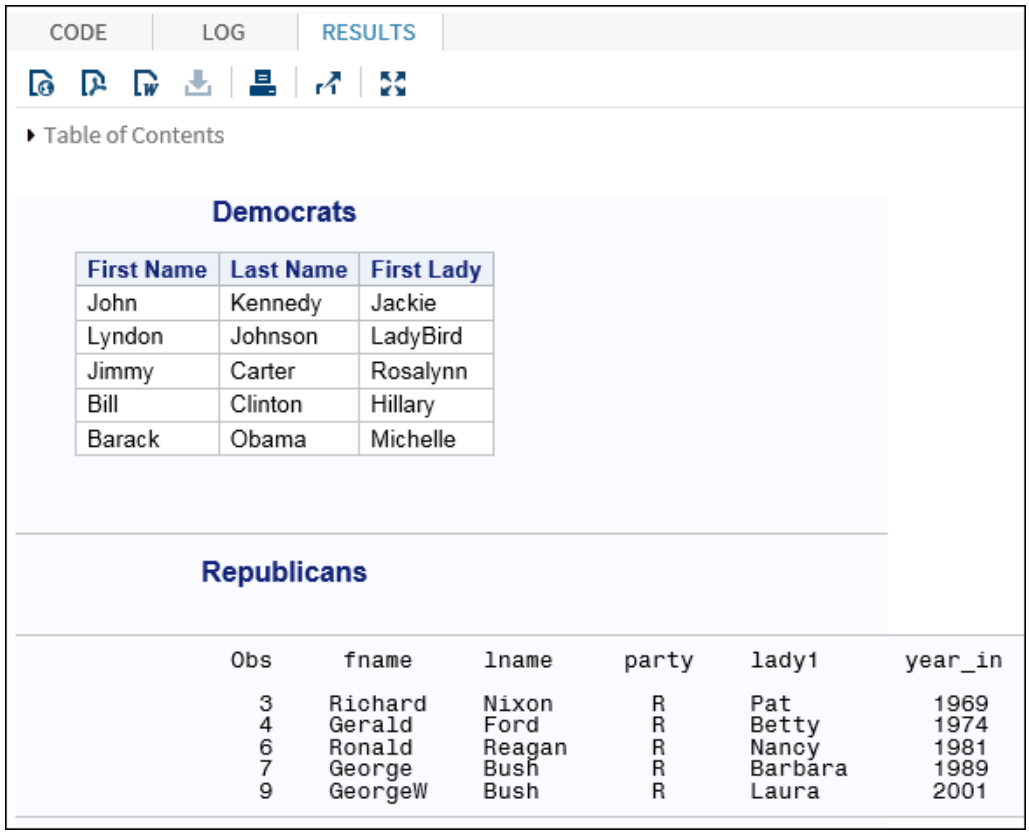
1 Create a data set on the server from data that is entered from the client.

2 Create a subsetted view on the server.

3 Use ODS to create an HTML table on the client using remote SQL PassThru to the SQL view on the server.

4 Use ODS to create a SAS data set.

Figure 3.4 *SAS Studio Results Tab*



Example 5: MP CONNECT and the WAITFOR Statement

Purpose

This example enables you to perform two encapsulated tasks in parallel, but both tasks must be completed before the client session can continue.

The following program sorts two data sets asynchronously. After both sort operations are complete, the results are merged.

Program

```
signon remote1 sascmd="!sascmd -nosyntaxcheck -noterminal";
 signon remote2 sascmd="!sascmd -nosyntaxcheck -noterminal";

rsubmit remote1 wait=no; 1
libname mydata '/project/test1';
   proc sort data=mydata.part1; 2
      by x;
   run;
endrsubmit;

rsubmit remote2 wait=no; 3
libname mydata '/project/test2';
   proc sort data=mydata.part2; 4
      by x;
   run;
endrsubmit;

waitfor _all_ remote1 remote2; 5

libname mydata ('/project/test1' '/project/test2'); 6
data work.sorted;
   merge mydata.part1 mydata.part2;
run;
```

1 Remote submit the first task.

2 Sort the first data set as one task. Because WAIT=NO, both tasks are processed at the same time.

3 Remote submit the second task.

4 Sort the second data set as one task.

5 Wait for both tasks to complete.

6 Merge the results and continue processing.

Example 6: MP CONNECT with Piping

Purpose

In this program, the MP CONNECT piping facility uses ports rather than disk devices for data I/O. The first process writes a data set to Pipe1. The second process reads the data set from Pipe1, performs a calculation, and directs final output to a disk device. The P1 and P2 processes execute asynchronously.

Program

```
signon p1 sascmd='!sascmd'; 1
rsubmit p1  wait=no;

libname outLib sasesock ":pipe1";

data outLib.Intermediate; 2
   do i=1 to 5;
       put 'Writing row ' i;
       output;
   end;
run;
endrsubmit;

signon p2 sascmd='!sascmd'; 3
rsubmit p2  wait=no;

libname inLib sasesock ":pipe1";
libname outLib "/tmp";

data outLib.Final;
set inLib.Intermediate;
   do j=1 to 5;
      put 'Adding data ' j;
      n2 = j*2;
      output;
   end;
run;
endrsubmit;
```

1 Process P1 in the first DATA step.

2 Create data set and write to pipe.

3 Process P2 in the second DATA step.

Example 7: Prevent Pipes from Closing Prematurely

Purpose

The TIMEOUT= option in the LIBNAME statement can be useful if a considerable delay is anticipated between the time that one task tries to read from a pipe and the time when another task starts to write to that pipe.

In this program, task P1 performs several DATA steps, a PROC SORT, and a PROC RANK, which is the step that writes to the pipe OUTLIB. However, task P2 is idle before the execution of the DATA step, which reads from the pipe INLIB. Therefore, a longer time-out is specified in the INLIB LIBNAME statement in order to allow sufficient time for task P1 to complete its processing before writing its output to the pipe.

Program

```
rsubmit p1 wait=no;
      libname outLib sasesock "pipe" timeout=10000;
      data a b;
         do i=1 to 10;
            output;
         end;
      run;
      data c;
         set a b;
      run;
      proc sort data=c out=sorted;
         by i;
      run;
      proc rank data=sorted out=outLib.ranked;
         var i;
         ranks Check;
      run;
   endrsubmit;
   rsubmit p2 wait=no;
      libname inLib  sasesock "pipe" timeout=60000;
      data fromPipe;
         set inLib.ranked;
      run;
      proc print; run;
   endrsubmit;
```

Example 8: Force Macro Variables to Be Defined When %SYSRPUT Executes

Purpose

In MP CONNECT processing, by default, macro variables in an RSUBMIT block are defined only when a synchronization point is encountered. In order to force macro variables to be defined when the %SYSRPUT macro variable executes, specify CSYSRPUTSYNC=YES in each RSUBMIT statement.

CAUTION

If the values that are specified in the CSYSRPUTSYNC= option differ between consecutive RSUBMIT blocks, the latter value supersedes the former value. If the SYSRPUTSYNC system option is specified, the CSYSRPUTSYNC= option in the RSUBMIT statement takes precedence. If the CSYSRPUTSYNC= option in an RSUBMIT block is omitted, the value for the system option is applied.

In the following program, the CSYSRPUTSYNC=YES option is specified in each RSUBMIT block in order to force macro variables to be defined for each %SYSRPUT macro variable execution. Without an explicit setting of CSYSRPUTSYNC=YES in each RSUBMIT block, a default value is provided by the SYSRPUTSYNC system option. The default is CSYSRPUTSYNC=NO, which causes macro variables to be defined when synchronization points are encountered.

Program

```
signon smp sascmd="!sascmd -logparm 'write=immediate' -nosyntaxcheck";
     options cwait=no;

/* -----------  first RSUBMIT block  ----- */
     rsubmit csysrputsync=yes;
       data a;
       do i=1 to 100;
       x=ranuni(0);
       output;
       end;
       run;

     %sysrput done=a;
     endrsubmit;

/* ----------- second RSUBMIT block  ----- */
     rsubmit csysrputsync=yes;
       data b;
       do i=1 to 100;
```

```
        x=ranuni(0);
        output;
        end;
        run;

      %sysrput done=b;
      endrsubmit;

/* ----------- third RSUBMIT block  ----- */
      rsubmit csysrputsync=yes;
        data c;
        do i=1 to 100;
        x=ranuni(0);
        output;
        end;
        run;

      %sysrput done=c;
      endrsubmit;

      waitfor smp;
      %put done=&done;
```

Example 9: Use Server Software from a Client Session

Purpose

Some software might not be available on each computer at your site. In addition, the software that is available on a server might perform some tasks better than the software that is available on your client. From a client session, you can use Compute Services to use software that is available on a server.

This program assumes that SAS/STAT is licensed only on the server. The program uses SAS/STAT to execute statistical procedures on the server.

Program: SAS/STAT Software

```
rsubmit;
      /************************************/
      /* The output from GLM is returned    */
      /* to the client SAS listing.         */
      /************************************/
   proc glm data=main.employee
      outstat=results;
      model sex=income;
   run;
```

```
      /*************************************/
      /* Use GLM's output data set RESULTS  */
      /* to create macro variables F_STAT   */
      /* and PROB, which contain the        */
      /* F-statistic PROB>F respectively.   */
      /*************************************/
   data _null_; set results
      (where=(_type_= 'SS1'));
      call symput('f_stat',f);
      call symput('prob',prob);
   run;

      /*************************************/
      /* Create macro variables that        */
      /* contain the two statistics of      */
      /* interest in the client session.    */
      /*************************************/
   %sysrput f_statistic=&f_stat;
   %sysrput probability=&prob;
endrsubmit;
```

Purpose

In the following example, because the server session has access to a fast sorting utility, it sorts the data and then transfers the sorted data to the client session.

Program: Sorting

```
rsubmit;
     /*************************************/
     /* Indicate to the server machine that*/
     /* the HOST sort utility should be    */
     /* used with PROC SORT. Ask SORT to   */
     /* subset out only those observations */
     /* of interest.                       */
     /*************************************/
  options sortpgm=host;
  proc sort data=tsolib.inventory
     out=out_of_stock;
     where status='Out-of-Stock';
     by orderdt stockid ;
  run;
     /*************************************/
     /* Output results; client will        */
     /* receive the listing from PRINT.    */
     /*************************************/
  title 'Inventory That Is Currently Out-
         of-Stock';
  title2 'by Reorder Date';
  proc print data=out_of_stock;
     by orderdt;
```

```
   run;
endrsubmit;
```

4

Using Remote Library Services (RLS)

Introduction to Remote Library Services

Definition

Remote Library Services (RLS) enables you to read, write, and update remote data as if it were stored on the client's disk. RLS can be used to access SAS data sets across computers that have different architectures. RLS also provides Read-Only access to some SAS catalog entry types across computers that have different architectures.

With RLS, you use a LIBNAME statement to associate a SAS library reference (libref) with a SAS library on the server.

Client Access to a Single- or Multi-User Server

To access a SAS library on a server that you are already signed on to (using the SIGNON statement), a single-user server environment is assumed. To identify the server, specify the remote session ID that was used at sign-on. For details about the SIGNON statement, see “SIGNON” on page 127.

To access a server that you are not signed on to, a multi-user environment is assumed. When you connect to a multi-user server, the server must already be running. Use the SERVER= option in the LIBNAME statement to specify the server ID.

Therefore, to connect to both a single-user server and a multi-user server from your client session, and to avoid confusion, assign unique values to the SERVER= option. The use of the single-user server takes precedence over the multi-user server.

After you define a libref to a server, avoid clearing and re-assigning the libref multiple times. Repeating this sequence is inefficient because the client session disconnects from the server after the last libref that is associated with a server is cleared. When the same libref is re-issued, the client session must connect to the

server again. To avoid this overhead, clear the defined librefs only after you have completed any processing that accesses data that is defined by these librefs.

A server does not automatically terminate after the last LIBNAME statement is cleared. A multi-user server remains active, awaiting connections from clients until the server administrator explicitly stops the server by using the PROC OPERATE statement. For more information, see "OPERATE Procedure" in *SAS/SHARE User's Guide*.

A single-user server remains active, awaiting connections from a client session until the client uses the SIGNOFF command to terminate the server session. For details, see "SIGNON" on page 127.

Advantages

If you need to maintain a single copy of the data on a server and keep the processing on the client, then RLS is the correct choice. In general, RLS is the best solution in the following situations:

- The amount of data that is needed by the client is small.
- The server data is frequently updated.
- Your data center rules prohibit multiple copies of data.

RLS enables you to access your server data as if it were local. This feature eliminates the explicit step of coding an upload or download of the data before processing it. It also permits the GUI of an application to reside at the client while the data remains at the server (for example, a client FSEDIT session of a server data set). You can build applications that provide seemingly identical access to client and server data, without requiring the end user to know where the data resides.

Using RLS, you can access and update data that is stored in an external database. RLS enables a client (single user) to access data that is stored in an external database and to update the data through the server (single user).

Considerations for Using RLS

Determine the Appropriate Data Access Solution

To make the best use of RLS, consider these questions:

- How much data will the application access?
- Is multi-user or single-user data access needed?
- Will the application make a single pass or multiple passes through the data?

- What is the effect of the application's data access on the network load?

Answers to these questions will help you determine whether to use RLS, Data Transfer Services, Compute Services, or a combination of these services.

Compute Services to Access Large Volumes of Data

Accessing data through RLS is inefficient when you have large volumes of data. Compute Services (or a combination of Compute Services and Data Transfer Services) is preferable for processing large volumes of data on the server.

Data Transfer Services for Multi-Pass Data Processing

RLS is not efficient for multiple passes through the data. Although the client accesses data that is on the server, the data is not written to the client's local disk. If you are running procedures that make multiple passes through the data, or an entire procedure must be run more than one time against the data, transferring a copy of the data to the client's local disk is advised. You incur the network traffic cost only one time rather than paying the cost for each pass through the data.

Data Transfer Services When Network Response Time Is Delayed

Data Transfer Services is the preferred choice when response time is delayed. This situation can occur if you are accessing server data that is being updated simultaneously by other users. If delayed response time is not acceptable, consider transferring a copy of the data to the client's local disk and keep the data separate from other applications.

RLS When Data Flow through a Network Is Minimal

Because RLS requires data to flow from the server to the client through a network, you should design your application to minimize the amount of data that is requested for client processing.

Both Data Transfer Services and RLS transfer data from the server to the client for processing. However, the difference between the two services is that Data Transfer Services writes the data to the client's local disk for subsequent processing. By contrast, RLS processes the data in client memory, which gets overwritten when the next data transaction occurs. Subsequent analyses of the same data would require

the data to be moved through the network each time the client session requests the data.

DTS, RLS, and CS Compared

Design your application to balance the benefits and costs of the SAS/CONNECT services.

- Use Data Transfer Services to transfer a copy of the data from the server to the client and write the data to disk for local data access and processing.
- Use Remote Library Services to transfer records that the client requests for processing from the server. All of the data remains at the server and selected records are transferred to the client for local processing.
- Use Compute Services to transfer processing to the server where the data is stored. Results from server processing are returned to the client.

RLS to Access Types of Data

RLS Support for Data Types

RLS supports access to the following types of data:

- SAS catalog*
- SAS data set and SAS utility file
- SAS view (DATA step, PROC SQL, and SAS/ACCESS views)
- SAS database (MDDB)
- External database (such as Oracle)

*Catalog update is not supported if the computers that the client and the server run on do not have compatible architectures.

Access a Catalog

In order for a client to use RLS to update a catalog on a server, the architectures of the computers on which the client and the server run must be compatible. If computer architectures are incompatible, the following error message is displayed:

```
ERROR: You cannot open catalog name through
       server ID because write access to
       catalogs is not supported when the user
       machine and server machine have different
```

```
data representations.
```

Access an External Database

RLS and a SAS/CONNECT single-user server support Update access to data that is stored in an external database. The SAS/ACCESS engines and the SQL engine recognize the single-user server as one user and therefore enable Update access for external database sources.

However, SAS/ACCESS engines and the SQL engines prohibit Update access to external database sources when using RLS and a multi-user server. Updating is prohibited because of the inability of a multi-user server or a database to detect and manage conflicting requests from multiple users. A detection facility is necessary in order to generate audit trails and to guarantee data integrity and security.

Access a SAS View

RLS supports access to SAS views, which include DATA step views, SAS/ACCESS views, and PROC SQL views.

When the server accesses the library that contains the SAS view, the view is interpreted at the server by default. The server loads and calls the engine that is appropriate to the SAS view to read and transform the underlying data. The processing that is required to generate the SAS view is performed at the server, and the resulting SAS view is transferred to the client with a minimum cost to the network. Client resources are not used to interpret the SAS view.

For all PROC SQL views or for any other type of SAS view that is processed between a client and a server whose computer architectures are compatible, the SAS view can be interpreted at the client. To interpret a SAS view at the client instead of at the server, set the RMTVIEW= option to NO in a LIBNAME statement. Here is an example:

```
libname payroll rmtview=no server=wntnode;
```

For DATA step views and SAS/ACCESS views, if the architectures of the computers that the client and the server run on are different, the views can be interpreted only at the server.

Access a SAS Utility File of Type PROGRAM or ACCESS

In order for a client to use RLS to access a SAS utility file of the type PROGRAM or ACCESS on a server, the architectures of the computers that the client and the server run on must be compatible. If computer architectures are incompatible, the following error message is displayed:

```
ERROR: You cannot open utility file name through
       server ID, because access to utility
```

```
files is not supported when the user machine
and server machine have different data
representations.
```

A SAS utility file of the type PROGRAM contains compiled DATA step code, which cannot be processed at the client. The DATA step can be executed at the server if the DATA step is referenced by a DATA step view that is interpreted at the server.

Use SAS Views with Servers

SAS/ACCESS Views, DATA Step Views, and PROC SQL Views

RLS can be used with three types of SAS views:

- SAS/ACCESS views
- DATA step views
- PROC SQL views

A SAS view contains no data, but describes other data. A SAS view is processed by an engine that reads the underlying data and uses the description to return the data in the requested form. This process is called view interpretation.

When the library that contains the SAS view is accessed through a server, the SAS view is interpreted in the server's session by default. This means that the engine is loaded and called by the server to read and transform the underlying data. Only a small amount of data is moved through the network, and the client processing is unaware that a SAS view is involved.

If the SAS view is a PROC SQL view or if the client and server computer architectures are the same, you can cause the SAS view to be interpreted in the client session. This is done by specifying RMTVIEW=NO in the LIBNAME statement that is used to define the server library. If the architectures are not the same, SAS/ACCESS views and DATA step views can be interpreted only in the server session.

Interpreting a SAS view as data can produce significant processing demands. When a SAS view is interpreted in the client session, that frequently means that a lot of data has to flow to the client session. This removes processing demands from the server session but increases network load.

Recommendations for PROC SQL Views

PROC SQL views are especially good candidates for interpretation in a server session under these conditions:

- The number of observations that are produced by the PROC SQL view is much smaller than the number of observations that are read by the PROC SQL view.
- The data sets that are read by the PROC SQL view are available to the server.
- The amount of processing that is necessary to build each observation is not large.

Conversely, PROC SQL views should be interpreted in the client session under the following conditions:

- The number of observations that are produced by the PROC SQL view is not appreciably smaller than the number of observations that are read by the PROC SQL view.
- Some of the data sets that are read by the PROC SQL view can be directly accessed by the client session.
- A large amount of processing must be performed by the PROC SQL view.

WHERE Processing to Reduce Network Traffic

When using RLS, one of the best ways to reduce the amount of data that needs to move through the network to the client session is to use WHERE statement processing whenever possible. When WHERE statements are used, the WHERE clause is passed to the server environment and interpreted. Only the data that meets the selection criteria is transferred to the client environment for processing.

If the data that you are accessing is stored in an external database, the WHERE statement is passed to the database and evaluated, if possible. If the database cannot complete the evaluation, the server completes it before returning any of the data to the client session. For examples of using the WHERE statement, see the following:

- “Example 2: Access Server Data By Using the WHERE Statement” on page 80,
- “Example 4: An SCL Program That Uses the WHERE Statement” on page 81,
- “Example 6: Subset Server Data for Client Processing and Display” on page 84.

Example 1: Access Server Data to Print a List of Reports

Purpose

This code shows a client that uses RLS to access a modest amount of data on a server in order to print a list of reports. RLS is a good solution for processing a small number of observations.

Program

```
libname vcl "/tmp/mylib"; 1
data vcl.request;
   report_name="January";
   copy='Y';
   output;
   report_name="February";
   copy='N';
   output;
   report_name="March";
   copy='Y';
   output;
run;

signon rmthost user='myuserid' password='mypassword';

libname public REMOTE '/tmp/mylib' server=rmthost; 2
   data _null_;
   set public.request;
   if (copy = "Y") then do;
      put "Report " report_name
          " has been requested";
      end;
   run;
```

1 Creates a data set in the user's home directory.

2 Defines a server library to a client session. The value for SERVER= is the same as the server session ID that is used in the SIGNON statement.

Example 2: Access Server Data By Using the WHERE Statement

Purpose

In this example, WHERE statement processing modifies the previous example in order to reduce the amount of data that is being requested and to reduce the network traffic. The WHERE statement filters only the relevant data for the client to process. A selective transfer is more efficient than moving every observation to the client to process and to check the COPY variable for a `Y` value.

Program

```
signon rmthost user='myuserid' password='mypassword';

libname public '/tmp/mylib' server=rmthost; 1

data _null_; 2
   set public.request;
   where copy = "Y";
   put "Report " report_name
       " has been requested";
  run;
```

1 Defines a server library to a client session.

2 Uses the WHERE statement to filter unneeded observations.

Example 3: Update Server Data

Purpose

This example enables you to take advantage of a mainframe's superior data handling and security features, while at the same time you work in a user-friendly GUI environment. RLS is used to update server data. This application of RLS

eliminates the need to transfer a disk copy of the data to the client session before processing the data. It also involves low-volume transaction processing.

Program

```
x mkdir Hr.Emp.Data; 1
libname hr 'Hr.Emp.Data';

data hr.employee;
   x=1;
run;

signon remos390 user='myuserid' password='mypassword';

libname rlib REMOTE 'Hr.Emp.Data' server=remos390; 2

proc fsedit data=rlib.employee; 3
run;

signoff remos390;
```

1. Creates the data set Hr.Emp.Data.
2. Defines the server session human resource library to the client session.
3. Executes a client FSEDIT to update the employee data set that is located on the z/OS computer.

Example 4: An SCL Program That Uses the WHERE Statement

Purpose

This example is an excerpt from an SCL program that uses RLS to query a remote reservation database. Reservations are selected based on the value that is stored in the variable RESNUM. The use of the WHERE clause in this example is important because the WHERE clause is applied in the server session before any data is transferred. As a result, only the observations that meet the criteria are moved to the client session.

This example is a good use of RLS because (as in the previous example) it involves transaction-type processing and enables the client GUI to be used for data entry on the selected observations in the database.

However, if you were to use the SCL LOCATEC function, every observation would be transferred to the client session and compared against the specified criteria. The

response time might be poor. These alternative programming choices emphasize the importance of being aware of the amount of data that the client session requests and minimizing this amount when using RLS.

Program

```
signon apex user='myuserid' password='mypassword';
libname master REMOTE "hq.prod.data" server=apex;

rdsid = open("master.reserv", 'u'); 1

wherecls="resnum=" || "'" || resnum || "'"; 2
   rc = where(rdsid, wherecls);
   call set(rdsid);
   rc = fetchobs(rdsid, 1);
signoff apex;
```

1 Opens the remote database.

2 Builds and applies the WHERE clause to accelerate retrieval.

Example 5: Update a Server Data Set By Applying a Client Transaction Data Set

Purpose

In client/server jobs where data must be kept current and the number of updates that you need to perform is small, RLS can be an effective solution. RLS enables you to perform a client update to a server data set.

This example creates a data set by remotely submitting a DATA step. Next, it creates a client transaction data set. Using RLS, it assigns a client libref to the server library. Finally, the program uses the client transactions to modify the server data set.

Program

```
%let rsession=unxhost;
signon rsession user='myuserid' password='mypassword';
   rsubmit;
    data sasuser.my_budget; 1
       length category $ 9;
```

```
      input category $ balance;
      format balance dollar10.2;
      datalines;
  utilities   500
  mortgage   8000
  telephone  1000
  food       3000
  run;

  endrsubmit;

data bills; 2
   length category $ 9;
   input category $ bill_amount;
   datalines;
utilities    45.83
mortgage    649.95
food         68.21
run;

libname rlslib slibref=sasuser server=rsession; 3

data rlslib.my_budget; 4
   modify rlslib.my_budget bills;
   by category;
   balance=balance-bill_amount;
run;

data _null_;
   set rlslib.my_budget; 5
   put 'Balance for ' category  @25
       'is: ' balance;
run;

signoff rsession; 6
```

1 Creates the master data set My_Budget in the library Sasuser in the server session.

2 Creates a client transaction data set Bills for updating the server data set My_Budget.

3 Assigns the client libref RlsLib to the library Sasuser in the server session.

4 Applies the transaction data set Bills to the server data set My_Budget.

5 Reviews the results. Three observations are updated.

6 Signs off the server. The libref RlsLib is deassigned as part of the sign-off processing.

Example 6: Subset Server Data for Client Processing and Display

Purpose

If the amount of data that is needed for a processing job is small, RLS is an efficient way to gather current data that is on a server for client processing and display. This program subsets the data on the server so that only the data that you need is transferred. This method saves computing resources on the server and reduces network traffic while it gives you access to the most current data.

In this example, a large reservations database is located on a server that runs under the UNIX operating environment. Several client procedures need to be run against a small subset of the data that is contained in the master reservations database. This situation is ideal for RLS.

The LIBNAME statement is issued in the client session to define the server library that contains the data set **Reservc**. The PROC SORT statement sorts the server data set and writes the subset data to the client disk.

The WHERE= and KEEP= options are specified in the PROC SORT statement to reduce the amount of data that moves through the network to the client session for processing. Only the data that meets the WHERE= and KEEP= criteria is moved across the network to the client session.

PROC SORT creates the subset data set in the client session and allows all subsequent processing to run in the client session without additional server CPU consumption. PROC SUMMARY and PROC REPORT summarize and format the client data. ODS is used to create an HTML file.

Program

```
signon srv1 user='myuserid' password='mypassword';
libname remlib '/u/user1/reservations' server=srv1; 1

proc sort data= 2
   remlib.reservc(keep=company origin
   where=(origin='ATLANTA'))
      out=tmp;
      by company;
run;

proc summary data=tmp 3
   vardef=n noprint;
```

```
   by company;
   output out=tmp2;
run;

ods html body="body.htm"; 4

proc report ls=74 ps=85 split= 5
   "/" HEADLINE HEADSKIP CENTER NOWD;
   column
      ("Totals" "" "" "" company _freq_);
   define company / group format=$40.
      width=40 spacing=2 left "Company";
   define _freq_ / sum width=14
      spacing=2 right "# Reservations";
   rbreak after /ol dul skip summarize
      color=cyan;
run;

ods html close;

signoff srv1;
```

1 Executes the LIBNAME statement in the client session to define the server library.

2 PROC SORT runs in the client session but accesses the server data set **Reservc**. A subset of **Reservc** is written to the client data set TMP. The WHERE= and KEEP= options are passed to the server session and evaluated there to minimize the amount of data that must move across the network.

3 Summarizes the client data set.

4 Creates an HTML file.

5 Creates a report using the client summary data set.

5

Using Data Transfer Services

Introduction to Data Transfer Services

Data Transfer Services offers the best solution for the transfer of SAS data and external files between a SAS/CONNECT client and a server.

Data Transfer Services is most useful for data exchanges between a client and a server that run different operating environments on incompatible computer architectures (for example, z/OS and Windows) or different SAS software releases (for example, SAS 8 and SAS 9). Data Transfer Services automatically translates the internal representations of character and numeric data between the client and the server computers.

Note: The translation algorithm was changed between SAS 6 and SAS 8 and later releases of SAS. See "File Format Translation Algorithms" on page 462.

You implement Data Transfer Services by using the UPLOAD and DOWNLOAD procedures. Before Data Transfer Services can be deployed, a client session must be connected to a server session (for example, by using the SIGNON statement).

Data Transfer Services: Advantages

Offloads Server Work

A major benefit of Data Transfer Services is the ability to offload work from a server to a client. A redistribution of workload boosts response time for production systems that run on servers. After the data is downloaded to the client, the client's processor performs all subsequent data access and processing.

Increases the Robustness of a Decision Support Environment

Moving a copy of the data to the client adds robustness to your decision support environment. In the case of a network failure that would temporarily eliminate access to the server's data, you can continue working with your client copy of the data.

Transfers Only Relevant Data

You can transfer only the data that you need by using WHERE processing or data set options (such as the OBS= option) or both to dynamically subset the data as it is being transferred to the client or the server. WHERE processing reduces network traffic and gives you only the data that is needed at the client or the server.

Supports the Model of a Centralized Control Point

Data Transfer Services supports the model of a centralized control point, such as a mainframe, which initiates communication to a network of workstations.

This model enables centralized distribution of data and applications. Automated jobs that can run during non-peak hours can distribute data and applications to multiple

computers that need the data and the applications for the next day's work. Similarly, jobs can be set up to query a network of workstations for the purpose of gathering data and storing it in a centralized repository.

Backs Up Client Data

Data Transfer Services facilitates data backup. Data and applications can be copied from a client that has limited memory resources to a server that has more memory resources. This provides a backup in case of loss on the client.

Balances Resources in an Application Development Environment

In a program development environment, programmers can use Data Transfer Services to make efficient use of network resources. In the early phase of program development, the programmer can use client resources for basic programming activities (such as editing, testing, and debugging) that do not demand high-performance computing resources. However, when program development demands a high-performance environment for testing or data access, the programmer might use Data Transfer Services to relocate the application to the environment that provides the needed resources.

The development environments at many computing installations often have a higher number of users who work on one system than on other systems. On the system with the heaviest load, response time, execution queues, and other performance factors are less efficient because so many people are running applications concurrently.

Using Data Transfer Services, you avoid contention for heavily used computer resources by creating and testing SAS programs on a less busy system (the client), and then transferring the fully developed and tested program to the heavily loaded system (the server).

Each time you execute a program at the client for testing purposes, you avoid adding to the load on the server. This convenient method can result in significant savings of server resources.

For example, suppose you are developing a SAS program that will run as a production program on the server. Your program analyzes data from a SAS data set that is located on the server and creates several reports from the analysis information. To run many tests of the program before it is final and to avoid the delays that result from server connections, create and store the SAS program on the client. Test the program by downloading the SAS data set that is being analyzed by the program, or test the program by using data that is stored on the client. After the program is complete and correct, upload the program file to the server.

Considerations for Using Data Transfer Services

Use Compute Services to Access Large Data Resources

Transferring a copy of the data to another file system creates multiple copies of the data. If the data that is stored on the server is updated frequently, keeping a local copy of the data that is reasonably current might be impossible. In addition, security restrictions at your site might prohibit multiple copies of the data. In this case, if the amount of data that is involved is large, consider using Compute Services instead.

Use Remote Library Services to Access Small to Medium Data Resources

If the client accesses a small to medium amount of data, Remote Library Services allows the processing to occur at the client, with data coming from the server as the execution requests it. If you use a GUI application to access data that requires transparent access to remote data, you might want to use Remote Library Services.

Use a Combination of Services

There might be situations in which a combination of services is the best choice. For a list of examples, see the examples sections in DOWNLOAD Procedure on page 255 and UPLOAD Procedure .

File Transfer Performance

Network File Compression

By default, SAS/CONNECT uses network file compression whenever a file is transferred between a client and a server by using the UPLOAD and DOWNLOAD procedures.

SAS/CONNECT 8.2 introduced a network file compression algorithm that significantly improved performance for large data transfers. A large transfer is defined as a file whose size is 32K bytes or larger. In general, the larger the file, the greater the potential for a performance gain.

The goal of network file compression is to reduce the number of buffers that must be sent when uploading and downloading files across a network. In order to reduce the number of buffers that are used, buffers are packed to capacity for each network transfer.

The algorithm uses run-length encoding and sliding window compression. Consecutive occurrences of a single byte are compressed by using run-length encoding, and patterns of characters are compressed by using a sliding window that stores an offset to the previously occurring pattern in the compressed data.

However, performance benefits that result from data compression depend on the data itself. For example, significant compression that yields a performance benefit is expected for data that contains a regularly repeating pattern. However, for data that does not contain a regularly repeating pattern, compression would not produce a significant performance benefit.

To take advantage of the compression algorithm, both the SAS/CONNECT client and the server must run SAS/CONNECT 8.2 or a later release of SAS software.

Data File Compression to Disk

By contrast, you can specify that a file be compressed when it is written to disk by using the COMPRESS= data set option. For more information, see *SAS Data Set Options: Reference*.

The following statements show how to specify that a data set should be compressed when it is uploaded to disk:

```
data tax01 (compress=yes);
proc upload data=state out=fed;
```

Note: If the COMPRESS=YES data set option is not specified, the data set is not compressed before it is uploaded.

At the client, the following tasks are implicitly performed:

- The engine decompresses the data set as it is read from disk.

- PROC UPLOAD compresses the observations in the data set as they are put into a buffer for transfer to the server.

At the server, the following tasks are implicitly performed:

- PROC UPLOAD receives the buffer and decompresses the data set so that the observations can be written.
- The engine writes the decompressed data set to disk.

Note: In order to write the compressed data set to disk, you have to specify the COMPRESS=YES data set option as an argument in the OUT= option. Here is an example:

```
proc upload data=state out=fed (compress=yes);
```

Transfer Status Window

The Transfer Status window displays information about the status of the download or upload operation. You can specify whether the Transfer Status window is displayed by specifying CONNECTSTATUS=YES | NO in any of the following contexts:

- “CONNECTSTATUS” on page 110
- CONNECTSTATUS= system option in the RSUBMIT statement on page 165
- CONNECTSTATUS= sytem option in the SIGNON statement on page 130
- CONNECTSTATUS= system option in the PROC DOWNLOAD statement on page 259
- CONNECTSTATUS= system option in the PROC UPLOAD statement on page 221

Because the Transfer Status Window displays the progress of the file transfer dynamically, the information in the window changes as the transfer progresses. The information in the display includes the following:

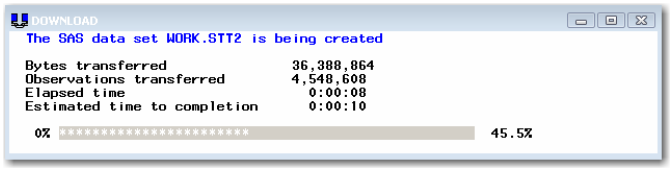
- the type of file that is being transferred (SAS data set, SAS catalog, catalog entry that contains graphics output, external file, or SAS utility file).
- the name of the target SAS data set, SAS catalog, external file, or SAS utility file. SAS data set names have the form *libref.SAS-data-set*. SAS catalog names have the form *libref.SAS-catalog*. External filenames are displayed with the complete filename. Utility filenames have the form *libref.SAS-utilityfilename*.
- the number of bytes being transferred (updated as each new buffer is sent).
- the number of observations being transferred (for SAS data sets only).
- the amount of time that elapsed since the beginning of the transfer, in *hh:mm:ss* form.
- an estimate of the amount of time that the transfer will take to complete, displayed as *hh:mm:ss*.

- the percentage of the file that has been transferred and a horizontal bar chart that depicts this percentage.

Note: For some types of files, the percentage completed, the estimated time to completion, and the bar chart are not always available. Some operating environments cannot efficiently provide the size of the file, which is necessary to calculate these estimates. Sometimes, the information that is provided by the operating environment results in estimates that are greater than the actual time that is needed for the transfer. Therefore, the percentage completed, the estimated time to completion, and the bar chart might show exaggerated estimates, but they will show 100% when the transfer is completed.

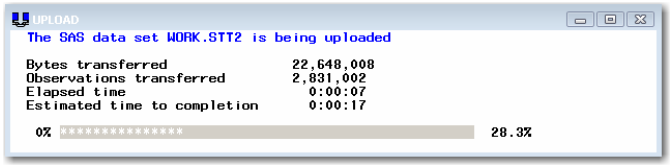
The following display is an example of the Transfer Status window during a SAS data set download. The SAS data set being downloaded is Work.Stt2.

Figure 5.1 *Transfer Status Window for Downloading a SAS Data Set*



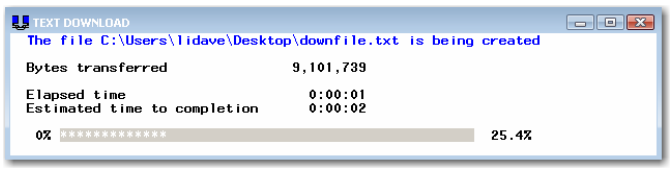
The following display is an example of the Transfer Status window during a SAS data set upload. The SAS data set being uploaded is Work.Stt2.

Figure 5.2 *Transfer Status Window for Uploading a SAS Data Set*



The following example shows the Transfer Status window when an external (flat) text file is being downloaded. The file being downloaded is `downfile.txt`.

Figure 5.3 *Transfer Status Window for Downloading an External File*



Non-English Keyboards

If you use a client that has a non-English keyboard, you probably have some external files that contain non-English characters. If your server runs under the z/OS

operating environment, some specially accented characters might be translated incorrectly when you use the DOWNLOAD and UPLOAD procedures. This occurs because of the default translations from ASCII to EBCDIC and from EBCDIC to ASCII. To solve the problem, you can do one of the following:

- If SAS/CONNECT is used frequently, you should use an alternate EBCDIC to ASCII translation table (TRANTAB=) on the server. Your SAS support personnel for the server should create the alternate table.
- If SAS/CONNECT is not used frequently, you can manage problematic characters by assigning the correct hexadecimal values in DATA step programming statements after the file is copied.

  For example, suppose you have a German keyboard and a z/OS operating environment. You want a file to contain A-umlaut characters after an upload. By default, the ASCII representation of A-umlaut, which is X'84', is translated to EBCDIC X'24'. However, the EBCDIC representation of A-umlaut is X'C0', so you need to translate EBCDIC X'24' to EBCDIC X'C0'. The following DATA step, in which NAME is a variable that contains A-umlaut characters, performs this translation:

```
data new;
   set old;
   retain to 'C0'x from '24'x;
   drop to from;
   name=translate(name,to,from);
run;
```

Data Transfer Services Tips

Tips for Using PROC DOWNLOAD and PROC UPLOAD

- To execute the DOWNLOAD and UPLOAD procedures in the server session, you must use the RSUBMIT command.
- The rate at which files are transferred varies according to these factors:
  - the size and number of files that are being transferred
  - the processing load on the server
  - the communication access method that is being used
  - the network configuration

  The Transfer Status window keeps you informed of the progress of the transfer. For details, see “Transfer Status Window” on page 92.
- You cannot transfer a SAS data set to an external file by using the DATA= or the INLIB= option.
- You cannot transfer an external file to a SAS data set by using the OUT= option.

- To transfer a text file whose record length is greater than 132 bytes, you must specify the LRECL= option in the FILENAME statement at both the client and the server. If you omit the LRECL= option, a data truncation error is reported. For details about the LRECL= option, see the FILENAME statement under "FILE Statement: z/OS" in *SAS Companion for z/OS*.

  Note: In SAS 9.4, the default value for LRECL is 32767. If you are using fixed length records (RECFM=F), the default value for LRECL is 256.

- If PROC DOWNLOAD or PROC UPLOAD successfully completes the file transfer, the macro variable SYSINFO is set to 0. If the file transfer is not completed successfully, the macro variable SYSINFO is set to a value greater than 0. You can pass the value of the SYSINFO macro variable back to the client by using the %SYSRPUT statement. For details, see "%SYSRPUT" on page 191.
- Statements that define librefs and filerefs in the client session must be executed in the client session by using the SUBMIT command.
- Statements that define librefs or filerefs in the server session must be executed in the server session by using the RSUBMIT command or the RSUBMIT statement. Therefore, if librefs or filerefs are defined before the PROC statement, these statements can be executed along with PROC DOWNLOAD or PROC UPLOAD.

Tips for Using PROC DOWNLOAD Only

- When downloading variable block records to a client from a server that is running under the z/OS environment, you must specify RECFM=U in the server FILENAME statement that points to the variable block record. For details about options in the FILENAME statement, see "FILENAME Statement: z/OS" in *SAS Companion for z/OS*.

  For example, if the file that you are downloading is called MYFILE, you would use the following code:

```
rsubmit;
   filename
      myfile 'vb.block.record' recfm=u;
   proc download infile=myfile
      outfile='c:\vb.rec' binary;
   run;
endrsubmit;
```

  After the client's Log window shows the number of bytes that are transferred, you would issue the following client FILENAME statement by using the RECFM= and LRECL= options, where the value of LRECL= is the number of bytes that were transferred:

```
filename myfile 'c:\vb.rec' recfm=s370vb
   lrecl=xxxx;
```

  The MYFILE fileref would then be used for subsequent access to the file.

Tips for Using PROC UPLOAD Only

- If you upload an external file to a server file that is defined with a fixed (F) record format, all records in the file are padded with blanks to the logical record length.

PART 3

SAS/CONNECT Language Reference

6

System Options

Dictionary

AUTOSIGNON System Option

Automatically signs on the client session to the server session, establishing a client/server connection when a connection does not already exist.

Client: Optional

Valid in: *Configuration file, OPTIONS statement, SAS System Options window, SAS invocation*

Category: Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Default: NOAUTOSIGNON

Syntax

AUTOSIGNON | **NOAUTOSIGNON**

Syntax Description

AUTOSIGNON
automatically signs on the client session to the server session for the subsequent execution of an RSUBMIT command or statement.

Note: In order to terminate a client/server session after an RSUBMIT has completed, you can do either of these:

- specify the NOCONNECTPERSIST system option
- issue an explicit SIGNOFF statement

NOAUTOSIGNON
does not automatically sign to the client session on the server session for the subsequent execution of an RSUBMIT command or statement. In order to establish a client/server connection, you must specify the SIGNON command or statement explicitly.

Details

When the AUTOSIGNON system option is specified, the RSUBMIT command or statement automatically executes a sign-on, and uses any SAS/CONNECT system options in addition to options that are specified in the RSUBMIT statement. For example, if you specify either the NOCONNECTWAIT system option or the NOCONNECTWAIT option in the RSUBMIT command or statement, asynchronous RSUBMITs will be the default for the entire connection.

For an example of using the AUTOSIGNON option with MP CONNECT, see “Example 5: MP CONNECT and the WAITFOR Statement” on page 63.

See Also

Statements:

- “RSUBMIT” on page 161

- “SIGNON” on page 127

System Options:

- “CONNECTPERSIST” on page 107

COMAMID= System Option

Identifies the primary communications access method to connect a client and a server across a network.

Client:	Optional
Server:	Optional
Valid in:	SAS/CONNECT Client: Configuration file, OPTIONS statement, SAS System Options window, SAS invocation, SAS/CONNECT Server: Configuration file, SAS invocation, SAS/SHARE Client and Server: Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Defaults:	TCP/IP for UNIX and Windows XMS for z/OS

Syntax

COMAMID=*access-method-ID*

Syntax Description

access-method-ID
specifies the name of the communications access method that is used by a client to access a server.

Details

You are not required to define TCP/IP as the communications access method under UNIX and Windows because TCP/IP is the default access method. However, you can choose to explicitly define TCP/IP as the access method. If you are running under z/OS and want to use TCP/IP instead of the default XMS, you must define TCP/IP as the access method.

If you are using SAS/SHARE under z/OS, the COMAUX1 option enables you to specify a primary and an alternative access method. For more information, see “COMAUX1= System Option” in *SAS/SHARE User’s Guide*.

Examples

Example 1: Use COMAMID in an OPTIONS Statement

The following OPTIONS statement specifies the TCP/IP access method for connecting to a server:

```
options comamid=tcp;
```

Example 2: Use COMAMID in a TYPE Statement with SAS/CONNECT

For SAS/CONNECT at the server, the TYPE statement in a script file specifies options that are set when the server session starts.

```
type "sas (dmr comamid=tcp noterminal no$syntaxcheck)" enter;
```

CONNECTEVENTS System Option

Specifies whether SAS events are propagated from the CONNECT server through the CONNECT client to SAS Enterprise Guide or to Add-in for Microsoft Office (AMO).

Client:	Optional
Server:	Optional
Valid in:	Configuration file, SAS invocation, OPTIONS statement, SAS system options window
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Requirements:	You must specify CONNECTEVENTS on both the CONNECT client and the CONNECT server for the events to propagate to SAS Enterprise Guide or to AMO. If a CONNECT client specifies NOCONNECTEVENTS, that CONNECT client will not receive events from the CONNECT server. Setting NOCONNECTEVENTS on the client stops events for that client. If NOCONNECTEVENTS is specified on a CONNECT server invocation, all CONNECT clients that use that server invocation will be prevented from receiving events.

Syntax

CONNECTEVENTS | **NOCONNECTEVENTS**

Syntax Description

CONNECTEVENTS
allows SAS events to be propagated from a CONNECT server through the CONNECT client to SAS Enterprise Guide or AMO.

NOCONNECTEVENTS
prevents the propagation of SAS events from a CONNECT server through the CONNECT client to SAS Enterprise Guide or AMO. The default setting is NOCONNECTEVENTS.

Details

You can use the CONNECTEVENTS | NOCONNECTEVENTS system option to specify whether to allow the propagation of SAS events from a CONNECT server through the CONNECT client to SAS Enterprise Guide or AMO.

This option can be set at start-up or anytime during the SAS Session. Your site administrator can restrict the modification of this option. NOCONNECTEVENTS is turned on by default.

See Also

Grid Computing

- *Grid Computing in SAS*

CONNECTMETACONNECTION System Option

Specifies whether a SAS/CONNECT server is authorized to access a SAS Metadata Server at server sign-on.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, SAS invocation, OPTIONS statement, SAS system options window
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	CMETACONNECTION
Requirement:	Grid sign-ons or sign-ons to a SAS/CONNECT server when there is a metadata connection on the client

Syntax

CONNECTMETACONNECTION | **NOCONNECTMETACONNECTION**

Syntax Description

CONNECTMETACONNECTION
allows a SAS/CONNECT server to access a SAS Metadata Server at server sign-on by providing a one-time supply of sign-on credentials. This option is on by default.

NOCONNECTMETACONNECTION
prevents the SAS/CONNECT server from automatically accessing the SAS Metadata Server via a one-time supply of credentials during sign-on. Instead, the SAS/CONNECT server must be a trusted peer of the SAS Metadata Server or the credentials must be hardcoded directly in the SAS code to be executed in the server session.

Details

When a SAS/CONNECT client session has an active metadata server connection and signs on to a SAS/CONNECT server, the server is automatically given access to the SAS Metadata Server for the duration of the SAS/CONNECT server session. The client queries the SAS Metadata Server for the following credentials, which are passed to the SAS/CONNECT server:

- SAS Metadata Server
- SAS Metadata Server port
- SAS Metadata Server user name
- SAS Metadata Server password (this is a special one-time use password and not the user's normal password)

Because these credentials are passed to the server, the server does not have to meet either of the following requirements:

- to be a trusted peer of the SAS Metadata Server
- to cause the credentials hardcoded in the SAS program to be executed in the server session

The SAS/CONNECT server uses the temporary credentials to remain connected to the SAS Metadata Server for the duration of the server session, rather than having to make multiple connections to the SAS Metadata Server. This option offers convenience and improves security. Because the option is on by default, it is not necessary to specify CONNECTMETACONNECTION in your SAS program. However, if you want to prevent the remote server from automatically connecting to the metadata server at sign-on, you must specify the NOCONNECTMETACONNECTION in the options statement. If you do this, you can still access the metadata server, but you must explicitly specify the user ID and password in the SAS code (RSUBMIT statement).

Note: If you specify credentials using SAS system options for metadata (for example, the METASERVER= or METAPORT= system options), these values take

precedence over any default values. For more information, see "Overview of System Options for Metadata" in *SAS Language Interfaces to Metadata*.

Examples

Example 1: Access Metadata Credentials for a Grid Execution

Here is an example of SAS code in which the CONNECTMETACONNECTION system is enabled. The grdsvc_enable() function specifies that all server sessions be enabled for a grid execution. Also, the SAS Application Server contains the definition for the logical grid server that manages the grid environment.

Note: The CONNECTMETACONNECTION option could be omitted because it is the default.

The AUTHDOMAIN= option in the LIBNAME statement specifies the name of the authentication domain, which is a metadata object that manages the credentials (user ID and password) that are associated with the specified domain. Specifying the authentication domain is a convenient way to obtain the metadata-based user credentials rather than having to explicitly supply them during server sign-on.

```
%put %sysfunc(grdsvc_enable(_ALL_, server=SASApp));
options CONNECTMETACONNECTION;
signon process=job1;
rsubmit;
libname mylib oracle authdomain=defaultAuth;
endrsubmit;
```

Example 2: Access Metadata Credentials for a Server Sign-on

In this example, the CONNECTMETACONNECTION option is used with the SIGNON statement and the SERVER= option:

```
options CONNECTMETACONNECTION;
signon process=job1 server=SASApp;
```

Example 3: Supply Explicit User Credentials for a Grid Execution

Here is an example in which NOCONNECTMETACONNECTION is used:

```
%put %sysfunc(grdsvc_enable(_ALL_, server=SASApp));
options NOCONNECTMETACONNECTION;
signon process=job1;
rsubmit;
libname mylib oracle user=tom password=apex;
```

```
endrsubmit;
```

The user ID and password are explicitly specified in SAS code in order to access the SAS Metadata Repository.

See Also

Statement

- “RSUBMIT” on page 161
- “SIGNON” on page 127

CONNECTOUTPUT= System Option

For a synchronous RSUBMIT, directs the server's output and log to the client session.

Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	COUTPUT
Default:	BUFFERED

Syntax

CONNECTOUTPUT=BUFFERED | IMMEDIATE

Syntax Description

BUFFERED
For a synchronous RSUBMIT, directs the server's output and log to the client session after the server's buffer is full. This is the default.

IMMEDIATE
For a synchronous RSUBMIT, directs the server's output and log as it is generated to the client session.

Details

When the CONNECTOUTPUT= option is specified, the synchronous RSUBMIT processing can be conveniently viewed from the client session as it occurs in the server session.

If buffered output is specified, the server output and log are sent to the client session after the server's buffer is full. If immediate output is specified, the output and log are sent to the client session as they are generated.

See Also

Statement

- “RSUBMIT” on page 161

CONNECTPERSIST System Option

Specifies whether a connection between a client and a server persists (continues) after the RSUBMIT has completed.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	CPERSIST
Default:	CONNECTPERSIST

Syntax

CONNECTPERSIST | **NOCONNECTPERSIST**

Syntax Description

CONNECTPERSIST
continues a client/server connection after the RSUBMIT (with or without automatic sign-on) has completed. The server is not automatically signed off (disconnected from) the client.

NOCONNECTPERSIST
discontinues a client/server connection after the RSUBMIT (with or without automatic sign on) has completed. The server is automatically signed off (disconnected from) the client.

Details

The CONNECTPERSIST option is most useful when automatic sign-on (specified by using the AUTOSIGNON option) is enabled.

A continued connection after the completion of a current RSUBMIT enables you to perform subsequent processing tasks within the same client/server session without having to sign on again. To terminate a persistent connection, you must perform an explicit SIGNOFF.

In addition to being a system option, CONNECTPERSIST can be set as an option in the RSUBMIT statement. The option in the RSUBMIT statement or command takes precedence over the system option.

See Also

Statement

- “AUTOSIGNON” on page 99

System Option

- “RSUBMIT” on page 161

CONNECTREMOTE= System Option

Identifies the server session that a SAS/CONNECT client connects to.

Client:	Required
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	CREMOTE=, REMOTE=, PROCESS=

Syntax

CONNECTREMOTE=*server-ID*

Syntax Description

server-ID
identifies the specific server session that the client connects to. This ID might correspond to the name of the machine that the client connects to. If connecting to a server session on a multiprocessor machine (that is, a machine that is equipped with SMP hardware), the ID can be a descriptive name that you assign to the session.

Details

In addition to being a system option, CONNECTREMOTE= can be set as an option in the RSUBMIT and SIGNON statements. The option in an RSUBMIT or SIGNON statement or command takes precedence over the system option.

Examples

Example 1: CONNECTREMOTE= in SIGNON

To sign on, include the port in the host macro variable named APEX:

```
%let apex=fully-qualified-machine-name spawner-port;
signon connectremote=apex user=userId password="password";
```

After a successful sign on, the CONNECTREMOTE option value is updated.

Example 2: CONNECTREMOTE= in RSUBMIT

In this example, it is assumed that signon to APEX is already done.

```
rsubmit connectremote=apex;
  code-to-submit;
endrsubmit;
```

After a successful RSUBMIT, the CONNECTREMOTE option value is updated.

See Also

Statements

- “RSUBMIT” on page 161
- “SIGNON” on page 127

CONNECTSTATUS System Option

Specifies the default setting for the display of the Transfer Status window.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	CSTATUS, STATUS
Default:	CONNECTSTATUS

Syntax

CONNECTSTATUS | **NOCONNECTSTATUS**

Syntax Description

CONNECTSTATUS
specifies that the Transfer Status window is displayed during file transfers.

NOCONNECTSTATUS
specifies that the Transfer Status window is not displayed during file transfers.

Details

For synchronous processing, the CONNECTSTATUS system option specifies whether the Transfer Status window is displayed during a PROC UPLOAD or a PROC DOWNLOAD. This system option can be overridden by specifying the CONNECTSTATUS= option in subsequent PROC UPLOAD, PROC DOWNLOAD, RSUBMIT, and SIGNON statements.

For asynchronous processing (NOCONNECTWAIT), the CONNECTSTATUS system option and the CONNECTSTATUS= option in a SIGNON statement are ignored. To enable the Transfer Status window for asynchronous processing, you must specify CONNECTSTATUS=YES in the PROC UPLOAD, PROC DOWNLOAD, or RSUBMIT statement.

See Also

Conceptual Information:

- “Transfer Status Window” on page 92

Statements

- “RSUBMIT” on page 161
- “SIGNON” on page 127

Procedures

- DOWNLOAD Procedure on page 255
- UPLOAD Procedure on page 217

CONNECTWAIT System Option

Specifies whether remote submits are executed synchronously or asynchronously.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	CWAIT
Default:	CONNECTWAIT

Syntax

CONNECTWAIT | **NOCONNECTWAIT**

Syntax Description

CONNECTWAIT
specifies that RSUBMIT statements are executed synchronously. *Synchronous processing* means that server processing must be completed before control is returned to the client session.

NOCONNECTWAIT
specifies that RSUBMIT statements are executed asynchronously. *Asynchronous processing* permits the client or multiple server processes to execute in parallel. Control is returned to the client session immediately after an

RSUBMIT begins execution to allow for continued processing in the client session or other server sessions.

Details

The CONNECTWAIT system option specifies whether remote submits are executed synchronously. The default setting can be overridden by setting the CONNECTWAIT= option in the SIGNON statement or in subsequent RSUBMIT statements. The option in the RSUBMIT or SIGNON statement or command takes precedence over the system option.

If NOCONNECTWAIT is specified, you might also want to specify the CMACVAR= option in the RSUBMIT statement. Setting CMACVAR= enables you to learn the status of the current asynchronous RSUBMIT (whether it has completed or is still in progress).

See Also

Statements

- “RSUBMIT” on page 161
- “SIGNON” on page 127

DMR System Option

invokes a server session.

Server:	Required
Valid in:	Configuration file, SAS invocation
Category:	Environment Control: Initialization and Operation
PROC OPTIONS GROUP=	Environment Control

Syntax

DMR

Details

The DMR system option must be specified either in the server CONFIG.SAS file or in the TYPE statement in a SAS/CONNECT script file that starts a SAS session. Alternatively, it executes by default when connecting to a spawner.

The server session receives input from the client session and sends log and output lines to the client's Log and Output windows or files.

SASCMD= System Option

Specifies the command that starts a server session on a symmetric multiprocessing (SMP) computer.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications

Syntax

UNIX, Windows:

SASCMD=<"*SAS-command* <*SAS-system-options*>" | "!SASCMD *SAS-system options*">

z/OS:

SASCMD=<":*SAS-system-options*" | "!SASCMD *SAS-system-options*">

Details

UNIX Specifics: under the UNIX operating environment, this command starts a server session on a multiprocessor computer. The TCP/IP access method is used to connect to the server session. !SASCMD specifies that the same SAS command that was used to invoke the client session should be used to invoke the server session. The SAS command can be specified with additional or overriding SAS system options.

z/OS Specifics: under the z/OS operating environment, this command starts a server session on a multiprocessor computer, and passes values for the following SAS system options to the server session: DMR, COMAMID=, REMOTE=, SASHELP=, SASMSG=, SASAUTOS=, and CONFIG=. You might also specify additional SAS system options to be passed to the server session. The XMS access method is used to connect to the server session. The `fork` command under UNIX is used to spawn an MVS BPX address space, which inherits the same STEPLIB and USERID as the client address space.

Windows Specifics: under the Windows operating environment, this command starts a server session on a multiprocessor computer. The TCP/IP access method is used to connect to the server session. !SASCMDspecifies that the same SAS command that was used to invoke the client session should be used to invoke the server session. The SAS command can be specified with additional or overriding SAS system options.

SASCMD= is most useful for starting multiple sessions to run asynchronously on multiprocessor computers. You can also use SASCMD= to develop an application on a single-processor computer that will be executed later on a multiprocessor computer.

In addition to being a system option, SASCMD= can be set as an option in the SIGNON and the RSUBMIT statements or commands. The option in an RSUBMIT or SIGNON statement or command takes precedence over the system option.

Examples

Example 1

The following OPTIONS statement invokes a SAS session.

```
options sascmd="sas";
```

Example 2

The following OPTIONS statement invokes a SAS session with options specified.

```
options sascmd="sas <options>";
```

Example 3

The following OPTIONS statement invokes a server session on a computer under the z/OS operating environment and sets the MEMSIZE= and NONUMBER options.

```
options sascmd=":memsize=64M nonumber";
```

Example 4

The following OPTIONS statement invokes a server session on a computer under the z/OS operating environment with no additional SAS options.

```
options sascmd="any-string";
```

Example 5

The following OPTIONS statement specifies a script file to invoke SAS.

```
options sascmd="mysas.bat";
```

For the preceding example, the following code is contained in the text file MYSAS.BAT.

```
cd "C:\Program Files\SAS System\9.0"
mkdir mywork
sas -nosyntaxcheck -work "mywork" %*
```

Note: The %* positional parameter enables you to specify additional SAS options when you invoke SAS.

When the SASCMD= option is executed, the MYSAS.BAT script is executed.

See Also

Statements

- “RSUBMIT” on page 161
- “SIGNON” on page 127

SASFRSCR System Option

Is a Read-Only option that contains the fileref that is generated by the SASSCRIPT= option.

Client:	Optional
Server:	Optional
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications

Syntax

SASFRSCR

Details

The SASFRSCR option is not explicitly specified. A value for SASFRSCR is generated only if SASSCRIPT is specified. You can read the value for this option in an application that is written in the SAS Component Language (SCL), which prompts a user for the correct SAS/CONNECT sign-on script.

SASSCRIPT= System Option

Specifies one or more locations for SAS/CONNECT server sign-on script files.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Default:	Varies by operating environment

Syntax

SASSCRIPT= "*dir-name*" | <"*dir-name-1*","*dir-name-2*…"> | "*fileref*" | <"*fileref-1*","*fileref-2*…">

Syntax Description

"dir-name"* | *fileref
specifies the name of one or more directories that contain SAS/CONNECT script files. Enclose the directory name in double or single quotation marks. The directory name can also be specified as a fileref.

UNIX specifics *sas-installation-directory*/misc/connect

Windows specifics *sas-installation-directory*\connect\saslink

z/OS specifics &prefix.CTMISC

Details

If the CSCRIPT= option is specified in the SIGNON statement and the specified script file is not located in the current directory, then the location that is specified in the SASSCRIPT= option is used to find the specified script file.

If quotation marks are omitted from the value, SAS can misinterpret the value as a physical filename and an error condition can result. Using quotation marks ensures that the value is correctly interpreted as a directory path.

The SASSCRIPT= option also enables you to find the location of a script file that has been configured as a property in the SAS Metadata Repository. The script path is among the properties of the SAS/CONNECT server component in the SAS Application Server that is stored in the SAS Metadata Repository.

Note: In order to obtain a script file path from the SAS Metadata Repository, you must have access to the repository. These SAS options can be used to configure access to the SAS Metadata Repository: METAAUTORESOURCES=, METACONNECT=, METAPASS=, METAPORT=, METAPROFILE=, METAPROTOCOL=, METAREPOSITORY=, METASERVER=, and METAUSER=.

Examples

Example 1: Assign the File Path to SASSCRIPT=

In this example, the SASSCRIPT= option is used to specify an alternative file path to scripts for server sign-ons under the Windows operating environment.

```
options sasscript= "c:\my\favorite\scripts";
```

After the SASSCRIPT= option has been specified, the script can be invoked as follows:

```
signon remhost cscript="myscr.scr";
```

When `myscr.scr` is not located in the default location, a search for the script will be made at the location that is specified in the SASSCRIPT= option.

Here is an example in the SAS log of the representation of the SASSCRIPT= option and the assigned value:

```
SASSCRIPT=("c:\my\favorite\scripts")
```

SAS encloses the quoted file path in parentheses.

Note: The SASSCRIPT= option is an alternative to the RLINK fileref that is used in the FILENAME statement for identifying the location of a script file.

Example 2: Assign a Fileref to SASSCRIPT=

In this example, a FILENAME statement is used to assign the filename TESTFILE to the fileref Pointer. The OPTIONS statement is used to assign the SASSCRIPT system option to the value Pointer, which is a fileref to the filename TESTFILE. The fileref is not enclosed in quotation marks.

```
filename pointer 'testfile';
options sasscript=pointer;
```

Example 3: Obtain the Script File Path from the SAS Metadata Repository

In this example, the path to the server sign-on script has been configured as a property in the SAS Metadata Repository. Here is the code to access the SAS Metadata Repository and to find out the script path:

```
options metaserver="max.apex.na.com";
signon serverv="SASApp";
```

The METASERVER= option is used to specify the fully qualified domain name of the computer on which the SAS Metadata Server runs. The SIGNON statement and the SERVERV= option are used to produce a list of the properties of the SAS/CONNECT server component in the SAS Application Server that is stored in a SAS Metadata Repository. The name of the SAS Application Server is "SASApp."

Here is an excerpt of the output that is sent to the SAS Log:

```
1    options metaserver="max.apex.na.com";
2    signon serverv="SASApp";
NOTE: Server=          SASApp - Connect Server
      Remote Session ID=     remhost
      ServerComponentID=     A5SXFC1R.AU000002
      Remote Host=           max.apex.na.com
      Communication Protocol=TCP
      Port=                  7551
 Scriptpath= F:\admin\work\favescript.scr
      AuthDomain= DefaultAuth
      Wait= Yes
      SignonWait= Yes
      Status= Yes
      Notify= No
```

Knowing the script path and the script name, in a client session, you can sign on to a server session. Here is an example:

```
options sasscript= "F:\admin\work";
signon remhost cscript="favescript.scr";
```

Here is an alternative way to sign on to a server session:

```
signon remhost cscript="F:\admin\work\favescript.scr";
```

See Also

Statements

- “RSUBMIT” on page 161
- “SIGNON” on page 127

SIGNONWAIT System Option

Specifies whether a SAS/CONNECT sign-on should be executed asynchronously or synchronously.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	CONNECTSWAIT, SWAIT
Default:	SIGNONWAIT

Syntax

SIGNONWAIT | **NOSIGNONWAIT**

Syntax Description

SIGNONWAIT
specifies that a SAS/CONNECT SIGNON statement will execute synchronously. *Synchronous processing* means that a sign-on to a server session must complete before control is returned to the client session.

NOSIGNONWAIT
specifies that a SAS/CONNECT SIGNON statement will execute asynchronously. *Asynchronous processing* permits sign-ons to multiple server sessions to execute in parallel. Control is returned to the client session immediately after a sign-on when NOSIGNONWAIT is specified.

Details

You can use NOSIGNONWAIT to start multiple server sessions in parallel. Parallelism reduces the total amount of time that would be used to start individual connections to server sessions. This time savings allows the client session to do other processing, such as submitting units of work remotely to a server session, as soon as sign-on is complete.

If NOSIGNONWAIT is specified, you might also want to specify the CMACVAR= option in the SIGNON statement. Setting CMACVAR= enables you to learn the status of the current asynchronous SIGNON (whether it has completed or is still in progress).

In addition to being a system option, SIGNONWAIT can be set as an option in the RSUBMIT and SIGNON statements. The option in the RSUBMIT or SIGNON statement or command takes precedence over the system option.

See Also

Statements

- “RSUBMIT” on page 161
- “SIGNON” on page 127

SYSRPUTSYNC System Option

Sets %SYSRPUT macro variables in the client session when the %SYSRPUT statements are executed rather than when a synchronization point is encountered.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Alias:	CSYSRPUTSYNC, NOCSYSRPUTSYNC
Default:	NOSYSRPUTSYNC

Syntax

SYSRPUTSYNC | **NOSYSRPUTSYNC**

Syntax Description

SYSRPUTSYNC
specifies that the client session's macro variables is updated when the client session receives the results of the server session's execution of the %SYSRPUT macro. The results are delivered in the form of a packet. Specifying YES does not mean that the client's macro variables are updated immediately after the server's execution of the %SYSRPUT macro variable. YES means that the client's macro variables are updated when the client receives the packet from the server. Therefore, the exact time that the client's macro variables are updated depends on the availability of the client to receive the packet. If the client is busy, the server waits until the client is ready to receive the packet.

NOSYSRPUTSYNC
specifies that the client session's macro variables are updated when a synchronization point is encountered.

Details

This option is useful only when executing an asynchronous RSUBMIT, which is enabled via these methods:

- NOCONNECTWAIT system option
- CONNECTWAIT=NO option in RSUBMIT
- CONNECTWAIT=NO option in SIGNON

In addition to being a system option, CSYSRPUTSYNC= can be specified as an option in the RSUBMIT statement. The CSYSRPUTSYNC= option in the RSUBMIT statement or command takes precedence over the system option.

By contrast, a synchronous RSUBMIT is enabled via these methods:

- CONNECTWAIT system option
- CONNECTWAIT=YES option in RSUBMIT
- CONNECTWAIT=YES option in SIGNON

A synchronous RSUBMIT causes macro variables to be updated when a synchronization point is encountered.

Note: You should not change the value of the SYSRPUTSYNC= option between consecutive asynchronous RSUBMIT statements. Changing SYSRPUTSYNC= between asynchronous RSUBMIT statements causes unpredictable results.

See Also

Conceptual information

- “Synchronization Points” on page 193

Statements

- “RSUBMIT” on page 161
- “SIGNON” on page 127

TBUFSIZE= System Option

Specifies the size of the buffer that is used by the SAS application layer for transferring data between a client and a server across a network.

Client:	Optional
Server:	Optional
Valid in:	Configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Category:	Communications: Networking and Encryption

PROC OPTIONS GROUP= Communications

Default: Varies by operating environment. Value is determined by the TCP stack on the host operating system.

Syntax

TBUFSIZE=*buffer-size-in-bytes*

Syntax Description

buffer-size-in-bytes
specifies the size of the buffer that SAS/CONNECT uses for transferring data.

Note *buffer-size-in-bytes* must be specified as a multiple of 1024 bytes. You can also specify the value in kilobytes using the format *n*K.

Details

The TBUFSIZE= option defines the buffer for the SAS application layer. The TCPMSGLEN= option defines another buffer for the SAS communications layer. For more information about TCPMSGLEN=, which is used only by the TCP/IP communications access method, see the topic that is appropriate to your operating environment in .

Table 6.1 *Summary of Attributes for the TBUFSIZE= and TCPMSGLEN= Options*

System Option	Controlling SAS Layer	Purpose of Buffer
TBUFSIZE=	SAS Application	SAS/CONNECT uses the buffer to transfer data to the communications layer.
TCPMSGLEN=	SAS Communications	The TCP/IP access method uses the buffer to transfer data to a client or a server.

The SAS application layer does the following:

1. packs and compresses data records into a buffer until all the data has been processed or the buffer is full.
2. sends a buffer to the communications layer. Unless it is explicitly set using the TBUFSIZE= or TCPMSGLEN= options, the default buffer size is determined by the TCP stack on the host operating system. SAS/CONNECT uses the default

TCP stack settings and auto tuning (if implemented on the stack) to ensure optimal network performance.

Using the TBUFSIZE= option to maximize buffer size for the SAS application layer reduces the number of calls that the application layer makes to the communications layer for a data transfer. A reduction of calls to the communications layer saves resources and improves operating environment and network performance. Other factors, such as the amount of data and the network bandwidth, must be considered to optimize buffer performance.

The SAS communications layer does the following:

1. receives a buffer from the SAS application layer.
2. sends a buffer to the client or to the server. Unless it is explicitly set using the TBUFSIZE= or TCPMSGLEN= options, the default buffer size is determined by the TCP stack on the host operating system. SAS/CONNECT uses the default TCP stack settings and auto tuning (if implemented on the stack) to ensure optimal network performance.

As with the TBUFSIZE= option, an optimal value assigned to TCPMSGLEN= can save resources and improve network performance. TCPMSGLEN= can be set to transfer the entire buffer that it receives or to divide the data into multiple transfers.

To change the size of the TCP buffer, the TCPMSGLEN= option is specified at both the client and the server. If the client and the server do not use identical values for TCPMSGLEN=, the smaller buffer size is used.

In addition to being a system option, TBUFSIZE= can be set as an option in the SIGNON statement. The option in the SIGNON statement or command takes precedence over the system option.

CAUTION
Do not specify the TBUFSIZE= option in the server session.

You should specify the TBUFSIZE= Option only in the Client Session. If you specify the TBUFSIZE= option in a remote SAS invocation that runs an AUTOEXEC file, the allocated buffers might be insufficient to complete the processing of the AUTOEXEC file. Although the client can successfully sign on to the server session, the error message that would alert you to insufficient buffers might not be written to the server log immediately. Instead, the error message would be logged following the client's next request for server processing.

Specify the TBUFSIZE= option in the SIGNON statement in the client session when signing on the server session.

Example

In the following OPTIONS statement, the TBUFSIZE= option is used to set the buffer size to 64K:

```
options tbufsize=65536;
signon;
```

Alternatively, you can specify `tbufsize=64k`.

See Also

Statement

- “SIGNON” on page 127

TCPLISTENTIME= System Option

Specifies the amount of time a SAS/CONNECT server listens for a client to connect before terminating the CONNECT server session.

Client:	Optional
Valid in:	Configuration file, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications
Default:	300

Syntax

TCPLISTENTIME=*listen-time-in-seconds* |MIN | MAX

Syntax Description

listen-time-in-seconds
Specifies the amount of time in seconds that a SAS/CONNECT server listens for a client to connect before terminating the session. *listen-time-in-seconds* is any nonnegative integer less than 601. A value of 0 means there is no time limit.

MIN
The minimum value is 0 (no time limit).

MAX
The maximum value is 600.

Details

The TCPLISTENTIME= option is a portable SAS system option that enables you to control idle and unresponsive sign-on connections. The option enables you to specify how long (in seconds) a server "listens" for a response from the client during sign-on before it exits automatically. The default value for the session time-out is 300. The maximum value is 600 seconds.

The following are examples of valid TCPLISTENTIME= values:

- TCPLISTENTIME=*MIN*
- TCPLISTENTIME=*1*
- TCPLISTENTIME= *90*
- TCPLISTENTIME=*MAX*

TCPPORTFIRST= System Option

Specifies the first value in a range of TCP/IP ports for a client to use to connect to a server.

Server:	Optional
Valid in:	Configuration file, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications

Syntax

TCPPORTFIRST=*n*

Syntax Description

n
specifies the first TCP/IP port in a range of ports for a client to use to connect to a server.

Details

Overview of the TCPPORTFIRST System Option

To assign the range of ports, assign the first port by using the TCPPORTFIRST= system option and the last port by using the TCPPORTLAST= system option. To restrict the connection to one port, specify the same value for both options. The TCPPORTFIRST= option is valid only in a SAS/CONNECT server session.

The TCPPORTFIRST / TCPPORTLAST on page 126 options can be specified on the SAS/CONNECT client and on the SAS/CONNECT server. The options control the TCP/IP listening port of the SAS/CONNECT client and server.

Operating Environment Information

Valid values for this option are specific to a given operating environment. For more information, see the SAS documentation for your operating environment, or contact your system administrator for information about valid values.

TCPPORTLAST= System Option

Specifies the last value in a range of TCP/IP ports for a client to use to connect to a server.

Server:	Optional
Valid in:	configuration file, SAS invocation
Category:	Communications: Networking and Encryption
PROC OPTIONS GROUP=	Communications

Syntax

TCPPORTLAST=*n*

Syntax Description

n
specifies the last TCP/IP port in a range of ports for a client to use to connect to a server.

Details

Overview of the TCPPORTLAST System Option

To assign the range of ports, assign the first port by using the TCPPORTFIRST= system option and the last port by using the TCPPORTLAST= system option. To restrict the connection to one port, specify the same value for both options. The TCPPORTLAST= option is valid only in a SAS/CONNECT server session.

Operating Environment Information

Valid values for this option are specific to a given operating environment. For more information, see the SAS documentation for your operating environment, or contact your system administrator for information about valid values.

7

SIGNON and SIGNOFF Statements

Dictionary

SIGNON Statement

Initiates a connection between a client session and a server session.

Valid in: client

Syntax

SIGNON <*options*>;

Optional Arguments

AUTHDOMAIN=*auth-domain* | "*auth-domain*"
specifies the name of an authentication domain, which is a metadata object that manages the credentials (user ID and password) that are associated with the specified domain. Specifying the authentication domain is a convenient way to obtain the metadata-based user credentials rather than having to explicitly supply them during server sign-on.

An administrator can define an authentication domain by using the User Manager in SAS Management Console.

Examples:

```
authdomain=DefaultAuth
authdomain="SAS/CONNECT Auth Domain"
```

Requirements The authentication domain and the associated credentials must be stored in a metadata repository, and the metadata server must be running in order to resolve the metadata object specification.

Enclose domain names that are not valid SAS names in double or single quotation marks.

Interaction If you specify AUTHDOMAIN=, do not also specify USERNAME= and PASSWORD=. Otherwise, sign-on is canceled.

See For complete details about creating and using authentication domains, see the *SAS Intelligence Platform: Security Administration Guide*.

SAS Management Console: Guide to Users and Permissions and SAS Management Console online Help.

CMACVAR=*value*

specifies the name of the macro variable in which SAS stores a code indicating the state of the current sign-on. When a SIGNON is executed, SAS checks the state of the sign-on and stores a return code of 0, 1, or 2 in the specified CMACVAR variable. The return code is generated after SIGNON processing is complete and the name that you specify becomes the default name for the current server session.. The CMACVAR macro variable can then be programmatically queried to learn the processing status of the sign-on (completed, failed, or in progress). See Table 7.4 on page 128 for a description of what each return code means.

Table 7.1 *CMACVAR Macro Variable Values in SIGNON*

Value	Description
0	The sign-on is complete.
1	The sign-on failed.
2	You have already signed on to the current server session.
3	The sign-on is in progress.

Note: If the SIGNON command or statement fails because of incorrect syntax, the macro variable is not set.

Alias MACVAR=

Interactions This default can be overridden only by specifying the CMACVAR= option in the RSUBMIT statement or command.

If SYSERR is being used and it is already set to 1012 due to a previous error in a SIGNON, RSUBMIT, or SIGNOFF statement, then it will not be reset to 0 after submitting a subsequent successful SIGNON, RSUBMIT, or SIGNOFF statement. Because SYSERR is reset only at step boundaries, you can reset its value by performing a valid DATA step or PROC step. For more information about the SYSERR automatic macro variable, see "SYSERR" in *SAS Macro Language: Reference*.

See CMACVAR= option on page 162 in the RSUBMIT statement

Example "Example 5: Use CMACVAR to Test for a Successful Sign-on " on page 146.

CONNECTREMOTE=<*server-ID*>

specifies the name of the server session that you want to sign on to. If only one session is active, connectremote=*server-ID* can be omitted. If multiple server sessions are active, omitting this option causes the program statements to be run in the most recently accessed server session. The current server session is identified by the value that is assigned to the CONNECTREMOTE system option.

You can specify *server-ID* using the following formats:

process-name

process-name is a descriptive name that you assign to the server session on a multiprocessor computer when the SASCMD= option is used.

See SASCMD= option on page 138

""!SASCMD"" on page 139

Example
```
signon connectremote=emp1 sascmd="!sascmd";
```

computer-name.port-name

computer-name is the name of a server, and *port-name* is the name of the port that the spawner service runs on. If the computer name is longer than eight characters, assign the computer name to a SAS macro variable and use the macro variable name as the server ID.

In this example, it is assumed that that the SAS Connect spawner was started with a service name and that name is being used.

Example
```
%let sashost=hrcomputer1.dorg.com;
signon connectremote=sashost.spawner-servicename
       user=userId password="password";
```

computer-name.port-number

computer-name is the name of a server, and *port-number* is the port that the spawner service runs on.

CAUTION

Specifying computer-name.port-number for the server ID will fail under these conditions:

- when used in a WAITFOR statement that is used to wait for the completion of an asynchronous RSUBMIT.

  Instead, use a one-level name, such as the *computer-with-port*

- when used in a LIBNAME statement.

  Instead, use a one-level name or a two-level name, such as *computer-name._ _port-number*.

Restriction Do not use this format as the value for the <*server-ID*> in the SIGNON statement if you are going to specify a LIBNAME statement on the server. Instead, use the <*computer-name._ _port-number*> format for the <*server-ID*> value in both the LIBNAME statement and the SIGNON statement.

Example
```
signon connectremote=hrcomp1.2267 user=userId password="password";
```

computer-with-port

computer-with-port is a macro variable that contains the name of a server and the port that the spawner service runs on, separated by one or more spaces. This specification is appropriate in cases where the *server-ID* must be specified as a one-level name.

Example
```
%let sashost=hrcomp1.dorg.com 2667;
signon connectremote=sashost user=userId password="password";
```

computer-name.__port-number

computer-name is the name of a server and *port-number* is the port that the spawner service runs on. This format should be used to specify the *server-ID* value for the SERVER= option in a LIBNAME statement.

See "LIBNAME" on page 205

Example
```
signon connectremote=hrcomp1.__2267 user=userId password="password";
libname myLib server=hrcomp1.__2267;
```

Alias CREMOTE=, PROCESS=, REMOTE=

CONNECTSTATUS=YES | NO

specifies whether the Transfer Status window is displayed for file transfers within the current server session.

Here are the values for this option:

YES

specifies that the Transfer Status window is displayed for file transfers within the current server session.

Alias Y

NO

specifies that the Transfer Status window is not displayed for file transfers within the current server session.

Alias N

If the CONNECTSTATUS= option is omitted from the SIGNON statement, its value is resolved as follows:

1 If the CONNECTSTATUS system option is specified, the value for the CONNECTSTATUS system option is used.

2 If the CONNECTSTATUS= option is specified in a subsequent RSUBMIT, PROC UPLOAD, or PROC DOWNLOAD statement, that value will override the default value of CONNECTSTATUS= option for SIGNON.

3 Otherwise, the default behavior occurs. The default for a synchronous RSUBMIT is YES, which displays the Transfer Status window. The default for an asynchronous RSUBMIT is NO, which does not display the Transfer Status window.

Alias CSTATUS=, STATUS=

Default YES for synchronous RSUBMITs. NO for asynchronous RSUBMITs.

See "Transfer Status Window" on page 92

"CONNECTSTATUS" on page 110

CONNECTWAIT=YES | NO

specifies whether RSUBMIT blocks execute synchronously or asynchronously. Synchronous RSUBMIT statements are executed sequentially. An RSUBMIT must be completed in the server session before control is returned to the client session.

For asynchronous RSUBMIT statements, you can execute tasks in multiple server sessions in parallel. Control is returned to the client session immediately after an RSUBMIT begins execution to allow continued execution in the client session and in other server sessions.

Here are the values for the CONNECTWAIT= option:

YES

specifies that the RSUBMIT blocks execute synchronously.

Alias Y

NO

specifies that the RSUBMIT blocks execute asynchronously.

Alias N

If the CONNECTWAIT= option in a SIGNON statement is omitted, the value for the CONNECTWAIT= option is resolved as follows:

1 If the CONNECTWAIT option is specified as an option in the RSUBMIT statement, then the value specified in the RSUBMIT statement is used.

2 If the CONNECTWAIT option is specified as a system option, then the value for the system option is used.

3 Otherwise, the default behavior, to execute synchronously, occurs.

Alias CWAIT=, WAIT=

Default YES

Interactions

If CONNECTWAIT=NO is specified, you might also specify the CMACVAR= option. CMACVAR= enables you to programmatically test the status of the current asynchronous RSUBMIT to find out whether the task has completed or is still in progress.

When %SYSRPUT executes within a synchronous RSUBMIT, the macro variable is defined to the client session as soon as it executes.

When %SYSRPUT is executed within an asynchronous RSUBMIT, the macro variable is defined in the client session when a synchronization point is encountered. To override this behavior, use the SYSRPUTSYNC= system option.

Note If CONNECTWAIT=NO is specified, an automatic sign-off will not occur unless CONNECTPERSIST=NO is also specified.

See "SYSRPUTSYNC" on page 120

"Synchronization Points" on page 193

"CONNECTWAIT" on page 111

CSCRIPT=*file-specification*
specifies the SAS/CONNECT script file to be used during sign-on.

When the SIGNON command executes, SAS log messages for the server session are displayed in the LOG window of the client session.

file-specification
specifies the location of the SAS/CONNECT script file.

Here are the values for *file-specification*:

"*filename*"
"*fully-qualified-filename*"
specifies the name of the script file or specifies the name of the script file along with its location (pathname). Enclose the filename and fully qualified filename in double or single quotation marks.

fileref
is the name of the reference file that is associated with the script file. A previously executed FILENAME statement must define the fileref.

If the fileref that you define for the script is the default fileref RLINK, you can omit this specification from the SIGNON command.

"*SASSCRIPT-specification*"
is the physical location of the SAS/CONNECT script file in the directory that is specified by the SASSCRIPT system option.

Alias SCRIPT=

Interactions If multiple CSCRIPT= options are specified, the last specification takes precedence.

When you use the CSCRIPT= option, do not also use the NOCSCRIPT option. If you use NOCSCRIPT and CSCRIPT=, sign-on is canceled.

See NOCSCRIPT option on page 135

"Synchronization Points" on page 193

"FILENAME" in *SAS Global Statements: Reference*

INHERITLIB=(*client-libref1*<=*server-libref1*> ... *client-librefn*<=*server-librefn*>)
enables libraries that are defined in the client session to be inherited by the server session for Read and Write access. Also, each client libref can be associated with a libref that is named differently in the server session. A space is used to separate each libref pair in a series, which is enclosed in parentheses.

Note: Because the SAS Work library cannot be reassigned in any SAS session, you cannot reassign it in the server session either.

Restrictions The INHERITLIB= option cannot refer to an SPD Engine library that was defined with the option TEMP= .

The INHERITLIB= option does not support libraries assigned with the SASESOCK engine.

The INHERITLIB= option is not supported in either the SIGNON or the RSUBMIT statements to start a secondary (nested) SAS/CONNECT session in a remote SAS/CONNECT server session. If you use the option this way, the secondary session will continue, but the INHERITLIB= option will be ignored.

Interactions If you use the INHERITLIB= option and the SASCMD= option when signing on to a server session, then the server session attempts to access the client library directly rather than to inherit access to the library via the client session. If the client session and the server session attempt to access the same file simultaneously, only one session is granted exclusive access to the file. The other session's access to the file is denied.

SAS/CONNECT does not support concurrent multi-user access to the same file. This functionality is supported by SAS/SHARE.

See SASCMD= on page 138

Example This example shows that the libref named Local in the client session is inherited for use in the server session:

```
signon job1 inheritlib=(local work=remote);
rsubmit;
   libname local list;
   libname remote list;
   data local.a;
   x=1;
   run;
endrsubmit;
```

LOG=KEEP | PURGE | *file-specification*< NEW >
OUTPUT=KEEP | PURGE | *file-specification*< NEW >
Used only when NOSIGNONWAIT is in effect, these options direct the SAS log or the SAS output that is generated by the current server session to the backing store or to a file specification. A *backing store* is a SAS utility file that is written to disk in the client SAS Work library.

Here are the values for these options:

KEEP
spools log or output lines, as applicable, to the backing store or to the computer on which the client session is running. The log or output lines can be retrieved using the RGET, RDISPLAY, RSUBMIT CONNECTWAIT=YES, or SIGNOFF statement. This is the default.

PURGE
deletes all the log or output lines that are generated by the current server session. PURGE is used to save disk resources. If you do not need the data, you can use PURGE to remove large volumes of log or output data that are written to the backing store.

***file-specification* < NEW >**
specifies a file as the destination for the log or output lines. The file is opened for output at the beginning of the asynchronous RSUBMIT and is closed at the end of the asynchronous RSUBMIT. After the current RSUBMIT has completed, subsequent RSUBMIT log or output lines can be appended to the preceding RSUBMIT destination file using the LOG= or OUTPUT= options.

Note: Directing output to the same file for multiple concurrent asynchronous RSUBMIT statements is not recommended.

Here are the values for this option:

"*filename* "
is the physical location of the SAS log file or the SAS output file. Enclose the filename in double or single quotation marks.

fileref
is a SAS name that is associated with the physical location of the SAS log file or the SAS output file.

NEW
specifies that a new file is to be opened for output. If the file already exists, then it is deleted and re-created. NEW is not the default.

If you specify the NEW option on the RSUBMIT LOG= statement and the MOD option in the FILENAME statement simultaneously, then the NEW option will be honored and the specified file will be opened for output rather than appended.

Default KEEP

Interactions Use the LOG= or OUTPUT= option only when the SIGNONWAIT=NO option or the NOSIGNONWAIT system option has been specified. Otherwise, the option is ignored and a WARNING is displayed in the log.

If you direct the log or output lines to a file and then use RGET or RDISPLAY to retrieve the contents of an empty backing store, then you will receive a message such as the following:

```
WARNING: The LOG option was used to file
log lines for the current SIGNON.
There are no log lines for RGET to process.
```

If you use both the asynchronous RSUBMIT and the PROC PRINTTO statements, then you might expect that the PROC PRINTTO statement causes data from the server session to be written to the file that is specified in the PROC PRINTTO

statement. However, because the asynchronous RSUBMIT and the PROC PRINTTO statements execute simultaneously, predicting which operation will complete first is impossible. The timing of the completions of these operations determines whether the results are written to the SIGNON log or to the PROC PRINTTO log. If PROC PRINTTO is used in this way, then the LOG= or the OUTPUT= option in the SIGNON statement is ignored, and no data is written to the backing store or to the specified file.

Note Do not simultaneously use the asynchronous RSUBMIT and the PROC PRINTTO statement and redirect output. Redirecting output by using a LOG= or an OUTPUT= option in the SIGNON statement and using a locally submitted PROC PRINTTO statement can cause unpredictable results.

See "SIGNONWAIT" on page 118

NOCSCRIPT
specifies that no SAS/CONNECT script file should be used for sign-on. NOCSCRIPT accelerates sign-on and conserves memory resources.

Alias NOSCRIPT

Interaction When you use NOCSCRIPT, do not also use SASCMD=, SERVER=, or CSCRIPT=. If you use NOCSCRIPT with SASCMD=, NOCSCRIPT is ignored. If you use NOCSCRIPT with SERVER= or CSCRIPT=, sign-on is canceled.

Tip NOCSCRIPT is useful if SASCMD= has been specified in a spawner invocation.

See "CSCRIPT=*file-specification*" on page 132

NOTIFY=YES | NO | "*e-mail-address*"
specifies whether to notify the user that an asynchronous RSUBMIT has completed. The notification can be in the form of a message window or an email message. The NOTIFY option is enabled only at sign-on and remains in effect for the duration of the server session.

Here are the values for this option:

YES
enables notification via a message window. Here is the format of the default message: **`Asynchronous task TASK1 has completed.`** TASK1 is the server ID. The message window does not interfere with any other task executions in progress. To acknowledge the message and to close the window, click **OK**.

Alias Y

Example Here is an example of enabling notification in a SIGNON statement:

```
options sascmd="!sascmd";
signon process1 wait=no notify=yes;
rsubmit;
   %put should get notification window;
endrsubmit;
```

NO
disables notification. This is the default.

Alias N

"*e-mail-address*"
enables notification via an email message, and specifies the email address of the recipient for the notification. Email addresses are limited to a maximum of 256 characters. Enclose the email address in double or single quotation marks. The message includes information about the total time that was used for the RSUBMIT. If the LOG= and OUTPUT= options are also specified in a SIGNON statement, the email message identifies the locations of the log file and output file.

Default NO

Restriction Notification occurs only for asynchronous RSUBMIT statements.

Interactions When you specify the NOTIFY="*e-mail-address*" option, you can also specify the SUBJECT="*subject-title*" option.

If NOTIFY=YES and the NOTERMINAL system option has been specified, the request for notification is ignored. This message is displayed:

```
WARNING: The NOTIFY option is valid
only if a TERMINAL is attached to this
SAS session. Option will be ignored.
```

However, notification can be directed to an email address, regardless of whether the TERMINAL or NOTERMINAL system option has been specified.

If NOTIFY="*e-mail address*" is specified, but the email message cannot be sent, notification will occur in the form of a message window, which is the action that occurs when NOTIFY=YES. This behavior assumes that the NOTERMINAL system option has not been specified.

Notification fails if NOTIFY=YES or NOTIFY="*e-mail address*" and you specify statements or commands (such as RGET or SIGNOFF) during the asynchronous RSUBMIT that change execution from asynchronous to synchronous mode.

If NOTIFY="*e-mail address*" is specified, the SAS system and the operating environment that the SAS system runs under must be configured to support email. Without appropriate configuration, your attempt to specify notification via email might fail. Contact your system administrator for details.

See CONNECTWAIT=NO option on page 131

AUTOSIGNON System Option on page 99

LOG= and OUTPUT= options on page 133

SUBJECT= option on page 142

SAS system options that support email configuration: EMAILHOST, EMAILPORT, and EMAILSY in *SAS System Options: Reference*.

PASSWORD=*password* | "*encoded-password*" | _PROMPT_
specifies the password to be used when connecting to a server. The operating environment that the server runs under can also affect password naming conventions. The value for *password* is replaced by Xs in the SAS log. To protect your password, use the security software at your site to limit access to the SAS program statements that create the server session.

password
specifies a user-supplied password that meets the following requirements:

- can be up to 256 characters in length.
- can contain uppercase and lowercase letters.
- can contain periods (.) and spaces.

See For more information about password and user-ID naming conventions, see "User ID and Password Naming Conventions" on page 144.

Example Here is an example that uses the PASSWORD= option in the SIGNON statement:

```
signon rhost password=abc.1235;
```

"encoded-password"
specifies an encoded password that was created using the PWENCODE procedure. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords. To obtain an encoded password, use the PWENCODE procedure and specify the clear-text password as the value for the IN= option in the PROC PWENCODE statement. To use the generated encrypted password in a SIGNON statement, specify the entire string, including the key, as the value for the PASSWORD= option.

Here is an example showing how to encrypt the text password "svrmach" using the PROC PWENCODE statement:

```
proc PWENCODE in="svrmach"  method=sas004;
run;
```

The METHOD= option specifies the type of encryption to be used, which in this example is AES encryption. The encrypted password is generated in the form *{key}encoded-password*. The key is used to decode the password. Here is the log output that is generated by this sample code:

```
1    proc PWENCODE in=XXXXXXXXX  method=sas004;
2    run;

{SAS004}D79E9A1821465E55C2AFF53FCABD37FC20538488398C2264

NOTE: PROCEDURE PWENCODE used (Total process time):
      real time           1.01 seconds
      cpu time            0.31 seconds
```

In the following example, the password that was generated by the sample code above is used with the PASSWORD= option to sign on:

```
signon rhost
```

```
password="{SAS004}D79E9A1821465E55C2AFF53FCABD37FC20538488398C2264";
```

Note: The encoded password is case-sensitive.

See "PWENCODE Procedure" in *Encryption in SAS*

PROMPT
causes SAS to prompt the user for a valid password. This value enforces security.

Alias PASSWD=, PASS=, PWD=, PW=

SASCMD="*SAS-command*" | SASCMD" | SASCMDV"
signs on to the server session on the same symmetric multiprocessing (SMP) computer that the client session is running on. This option is most useful when client and server sessions run on SMP hardware.

"*SAS-command*"
specifies a user-defined command that is used to start a SAS process. SAS/CONNECT adds the proper options to make the SAS session a SAS/CONNECT server session. The command file that starts the SAS session is specific to your operating environment. File extensions vary according to operating environment. Windows filenames use `.bat` and `.cmd` as file extensions. UNIX extensions include `.sh`, `.csh`, and `.ksh`.

z/OS Specifics: The SASCMD= option does not support z/OS command files, so a z/OS host command file cannot be used as the value for the SASCMD option. However, you can specify SAS invocation options using the SASCMD option. To do this, use a colon followed by the desired options as shown in the following example:

```
sascmd=":ls=256"
```

Windows Specifics: On Windows, the TCP/IP access method appends the -COMAMID tcp, -ICON, -NOSPLASH, and -NOTERMINAL options

- Windows example:

```
signon session1 sascmd="c:\Program Files\SASHome\SASFoundation\9.4\sas";
```

- UNIX example:

```
signon session1 sascmd="sas -nosyntaxcheck";
```

- z/OS example:

  Because the SASCMD option does not support z/OS command files, a z/OS host command file is not specified as the value for the SASCMD option. However, SAS invocation options can be specified using the SASCMD option. To do this, use a colon (:) followed by the desired options:

```
sascmd=":ls=256"
```

  For more information about SASCMD sign-ons in a z/OS operating environment, see "MP Connections on z/OS" on page 374.

Note: Commands that contain spaces must be enclosed in double quotation marks.

The TCP/IP access method automatically adds options, such as -DMR, to the server session's SAS command.

Interactions the SASCMD= option that is specified in the SIGNON statement takes precedence over the SASCMD= system option.

When you use SASCMD=, do not also use NOCSCRIPT. Otherwise, NOCSCRIPT is ignored.

See “SASCMD=” on page 113

"!SASCMD"
signs on to a server session using the same command that was used to start the client session. For example, if the SAS client session was started using the command

```
sas -memsize 1024
```

then specifying "`!sascmd`" as the value for the SASCMD= option in a server sign-on causes the server session to be started using "sas" as the start-up command and –MEMSIZE as the start-up option.

There are some SAS invocation options that are not passed to the server session when the "!SASCMD" value is specified. In SAS 9.4M3 and later releases, the METAUSER, METAPASS, and LOGCONFIGLOC options are not passed to server sign-on sessions that are created using the "`!SASCMD`" value.

For example, if you started the SAS client session using the command

```
sas -memsize 1024 -metapass xyz -metauser abc
```

and you perform a server sign-on by specifying

```
signon session1 sascmd="!sascmd -tbufsize 2048"
```

then the only options that will be effective in the server sign-on are the -MEMSIZE and -TBUFSIZE options.

"!SASCMDV"
signs on to a server session using the same command that was used to start the client session and writes the SAS invocation to the SAS log. The “!SASCMDV” value is identical to the “!SASCMD” option value except that it also writes the SAS invocation to the SAS log. Here is an example showing the SASCMDV option specified in the SIGNON statement:

SERVER="*SAS-application-server*"
in a SAS Intelligence Platform deployment, specifies the name of a SAS Application Server that contains a SAS/CONNECT server component in its grouping. The SAS Application Server has been defined in the SAS Metadata Repository using SAS Deployment Wizard. The SAS Application Server is configured using a set of system resources, including a SAS/CONNECT server component and properties that start a SAS/CONNECT server session. The server properties are equivalent to the options that can be specified in the SIGNON statement.

"SAS-application-server"
specifies a SAS Application Server that contains a SAS/CONNECT server component, which has been defined in a SAS Metadata Repository.

When you use the SERVER= option, certain system resources must be configured before you can access a SAS Metadata Server. For details, see “Metadata Server-based Sign-ons” on page 18.

Requirements Enclose the name of the SAS Application Server in double or single quotation marks.

If the specified SAS Application Server does not contain a SAS/CONNECT server component, the server sign-on fails.

Interactions When you use SERVER=, do not specify any other options in the SIGNON statement. If other options are specified, sign-on is canceled and this message is displayed:

```
ERROR: Additional options are not valid
with the SERVER option on the SIGNON command.
These options should be specified in the server definition.
```

See “SERVERV="*SAS-application-server*" | _ALL_” on page 140

SAS Management Console: Guide to Users and Permissions and SAS Management Console online Help

SERVERV="*SAS-application-server*" | _ALL_

displays a verbose list of the properties that specify a SAS/CONNECT server sign-on. The server sign-on properties are equivalent to the options that can be specified in the SIGNON statement. The sign-on properties are associated with a SAS/CONNECT component, which is included in a set of system resources for the SAS Application Server.

When you use the SERVERV= option, certain system resources must be configured before you can access a SAS Metadata Server. Also, one or more SAS Application Servers should be configured and should contain one or more SAS/CONNECT components. For details, see “Metadata Server-based Sign-ons” on page 18.

"*SAS-application-server*"
specifies a SAS Application Server that contains a SAS/CONNECT server component, which has been defined in a SAS Metadata Repository.

ALL
displays the sign-on properties for all SAS Application Servers that have been defined in the SAS Metadata Repository.

Here is an example that displays the values for the SAS/CONNECT component that is contained in the SAS Application Server `SASApp`.

```
signon serverv="sasmain";
```

Here is the output:

```
Server=                 hrmach1 — SAS/CONNECT Server
Remote Session ID=      sashost
ServerComponentID=      A5Z3NRQF.AR00005L
Remote Host=            hrmach1.dorg.com
Communication Protocol= TCP
Service/Port=           sasconnect
Port=                   2267
Scriptpath=             tcpunix.scr
Tbufsize=               4096
Wait=                   No
```

```
SignonWait=             No
Status=                 No
Notify=                 "joe@apex.com"
Subject=                "hrmach1 task completed"
```

Requirement Enclose the name of the SAS Application Server in double or single quotation marks.

Interactions When you use SERVERV=, do not specify any other options in the SIGNON statement. If other options are specified, sign-on is canceled and this message is displayed:

```
ERROR: Additional options are not
valid with the SERVERV option on the
SIGNON command. These options should be specified
in the server definition.
```

See *SAS Management Console: Guide to Users and Permissions* and SAS Management Console online Help

SIGNONWAIT=YES | NO

specifies whether a sign-on to a server session is to be executed synchronously or asynchronously.

YES

specifies synchronous sign-on. A synchronous sign-on causes the client session to wait until the sign-on to a server session has completed before control is returned to the client session for continued execution. YES is the default.

Alias Y

NO

specifies an asynchronous sign-on. An asynchronous sign-on to a server session begins execution and control is returned to the client session immediately for continued execution. Asynchronous sign-on allows multiple tasks (including other sign-ons) to be executed in parallel. Asynchronous sign-ons reduce the total amount of time that would be used to execute individual sign-ons to multiple server sessions. Using the saved time, the client session can execute more statements.

Alias N

Default YES

Interactions The SIGNONWAIT= option in the SIGNON statement takes precedence over the SIGNONWAIT system option.

If SIGNONWAIT is specified as a system option and SIGNONWAIT= is not specified as an option in the SIGNON statement, then the system option will apply to the SIGNON statement.

If SIGNONWAIT= NO is specified, then the USERID= and PASSWORD= options cannot be set to _PROMPT_.

Tip To find out if sign-on has completed, use the LISTTASK statement or check the value of the macro variable specified on the CMACVAR= option in the SIGNON statement.

See “CMACVAR=*value*” on page 128

“LISTTASK” on page 197

SUBJECT="*subject-title*"
specifies the subject title for the email notification message that is sent after an asynchronous RSUBMIT completes. A subject title is limited to a maximum of 256 characters.

Here is an example showing how to specify a subject using email notification:

```
options remote=myhost sascmd="!sascmd";
signon notify="joe.smith@apex.com" subject="First task completed on &SYSHOSTNAME";
rsubmit wait=no;
   code-to-be-executed
endrsubmit;
```

Restriction If NOTIFY="*e-mail-address*" is not specified, SUBJECT= will be ignored.

Interactions If SUBJECT= is specified in the SIGNON statement, then the subject title will be used in email notifications for asynchronous RSUBMIT statements unless the SUBJECT= option is specified in the RSUBMIT statement.

If no SUBJECT= is specified, then the following default subject title is used:

```
SAS/CONNECT task TASK1 has completed.
```

TASK1 is the server ID.

See “NOTIFY=YES | NO | "*e-mail-address*"” on page 135

“RSUBMIT” on page 161

TBUFSIZE=*buffer-size-in-bytes*
specifies the size of the buffer that SAS/CONNECT uses for transferring data between a client session and a server session.

buffer-size-in-bytes
specifies the size of the buffer that SAS/CONNECT uses for transferring data. The value must be a number whose value is greater than 0 and is a multiple of 1024.

Default 32768 bytes

Interactions The TBUFSIZE= option in the SIGNON statement takes precedence over the TBUFSIZE= system option.

If TBUFSIZE= is specified as a system option in the client session and in the server session, the value in the client session takes precedence.

If TBUFSIZE= is specified as a system option in the client session but is not specified in the SIGNON statement, the system option value will be used.

Do not specify TBUFSIZE= system option in the server session. If the TBUFSIZE= system option is included in the server's SAS invocation, then an update to the server log might be delayed until the next client request for server processing has completed.

See "TBUFSIZE=" on page 121

USERNAME=*user-ID* | _PROMPT_
specifies the user ID to be used when connecting to a server session. Here are the values that can be assigned to USERNAME=:

user-ID
specifies the name to be used when signing on. For details about a valid user ID, see "User ID and Password Naming Conventions" on page 144.

PROMPT
specifies that SAS prompt the user for a valid user ID. This value enforces security.

Alias USER=, USERID=, UID=

Details

Difference between the SIGNON Command and Statement

The primary difference between the command and the statement is that the SIGNON command can be issued only from the command line in any client SAS windowing environment window or in a DM statement. The SIGNON statement must be followed by a semicolon (;) and can be used in any client session.

Difference between Synchronous and Asynchronous SIGNONs

A sign-on is executed either synchronously or asynchronously.

synchronous
Client session control is not regained until after the sign-on has completed. Synchronous processing is the default processing mode.

asynchronous
Client session control is regained immediately after the client issues the SIGNON statement. Subsequent programs can execute in the client session and in the server sessions while a sign-on is in progress.

Synchronous sign-ons display results and output in the client session. If the SIGNON is asynchronous, you can use the RGET and RDISPLAY commands and statements and the LOG= and OUTPUT= options to retrieve and view the results.

Difference between SIGNON and AUTOSIGNON

You can explicitly execute the SIGNON statement to establish a connection between the client session and the server session. A sign-on entails accessing the computer that the server session will run on and then invoking a SAS/CONNECT server session.

An automatic sign-on is an implicit sign-on to the server when the client issues a remote submit request for server processing. When the AUTOSIGNON system option is set, the RSUBMIT command or statement automatically executes a sign-on and uses any SAS/CONNECT system options in addition to any connection options that are specified with RSUBMIT. For example, if you specify either the NOCONNECTWAIT system option or the CONNECTWAIT=NO option in the RSUBMIT command or statement, asynchronous RSUBMIT command or statements will be the default for the entire connection.

User ID and Password Naming Conventions

Each user ID and password is limited to 256 characters that follow these conventions:

- Mixed case is allowed.

```
user=joe password=Born2run;
```

- Periods (.) and spaces are allowed.
- A null value, which is no value, that is delimited with contiguous quotation marks is allowed.

```
user=joe password='';
```

- Quotation marks must enclose values that contain one or more spaces.

```
user='joe black' password='Born 2 run';
```

- Quotation marks must enclose values that contain one or more special characters.

```
 user='joe?black' password='Born 2 run';
```

- Quotation marks must enclose values that contain one or more quotation marks.

```
user='"happy joe"' pw=_prompt_;
```

- Quotation marks must enclose values that begin with a numeric value.

```
user='apexdomain\joe' password='2bornot2b';
```

- Quotation marks must enclose values that do not conform to rules for user-supplied SAS names. For details about rules, see "Rules for User-Supplied SAS Names" in *SAS Language Reference: Concepts*.

z/OS Specifics: SAS/CONNECT supports passwords that have mixed case on z/OS, and it supports the IBM standard for password phrases that have a length of up to 100 characters. For information about the IBM standard for password phrases, see Allowing Mixed-Case Passwords (IBM) and Assigning password phrases (IBM).

Examples

Example 1: Sign On Using a SAS/CONNECT Script

The %LET macro statement stores the remote host name and port number in the macro variable **rhost**. The OPTIONS statement specifies the server-ID **rhost**, and the FILENAME statement identifies the SAS/CONNECT sign-on script. The SIGNON statement initiates the connection. The TCP/IP access method is assumed by default.

```
%let rhost=rcomputer1.dorg.com 7551;
options remote=rhost;
filename rlink 'external-file-name';
signon;
```

Example 2: Secured Sign-on Using an Encoded Password

The USERNAME= and PASSWORD=options in a SIGNON statement ensure a secured sign-on. At sign-on, the user is prompted for a user name and password, which is automatically supplied in its encoded form. For details, see the PASSWORD= option on page 137.

```
signon rhost
  password="{SAS004}AC8E81601E4BAC347510EEA3ADDDE43F96C21DB9F114A691";
```

Example 3: Create a Sign-on Windows Command File

If you use MP CONNECT, you might want each server session to execute on a different disk. You can use the SASCMD= option to specify a command file that contains a command to change to a specific disk for the server session to run on. An example follows of creating a Windows script named `mysas.bat`

```
set userdrive=%1
%userdrive%
mkdir \sassdir
cd \sassdir
"C:\Program Files\SASHome\SASFoundation\9.4\sas" -nosyntaxcheck
-work "mywork" %*
```

To execute the command file, specify its name as the value for SASCMD=.

```
signon sascmd="mysas.bat";
```

Example 4: Sign On to Two Server Sessions for Remote Processing

You want to run SAS programs on two server sessions and download data to your client session. The configuration follows:

- The client session runs under UNIX.

- A server session named TSO runs under z/OS.

From the client session, you can submit the following program from the Program Editor window in interactive or non-interactive line mode:

```
options comamid=tcp;
%let wnt=xyz.mydomain.com 7551;
1    signon wnt;

/*****************************************/
/* initiates connection to a z/OS server host  */
/*****************************************/
2   filename tsoscr '!sasroot/misc/connect/tcptso9.scr';
signon tso cscript=tsoscr;

3   /*****************************************/
/* submit statements to a Windows server    */
/*****************************************/
rsubmit wnt wait=no;
  statements to be processed by Windows server
endrsubmit;
4   /*****************************************/
/* submit statements to z/OS server */
/*****************************************/
rsubmit tso wait=no;
   statements to be processed by z/OS server
endrsubmit;
5   waitfor _ALL_ wnt tso;
/*****************************************/
/* ends both connections                 */
/*****************************************/
6   signoff tso cscript=tsoscr;
signoff wnt;
```

1 The client signs on to the server session WNT.

2 The client uses a SAS/CONNECT script to sign on to the server session TSO.

3 The WNT server session asynchronously processes the statements that are enclosed by the RSUBMIT and ENDRSUBMIT statements.

4 The TSO server session asynchronously processes the statements that are enclosed by the RSUBMIT and ENDRSUBMIT statements.

5 The client session waits for both RSUBMIT statements to complete.

6 The client uses scripts to sign off from both server sessions.

Example 5: Use CMACVAR to Test for a Successful Sign-on

The following example illustrates that the macro variable from a successful sign-on will be used if an unsuccessful attempt is made.

```
/*****************************************/
/* signon successful, rhost1 will be      */
/* set to 0 to indicate success.          */
/*****************************************/
```

```
signon rhost macvar=rhost1;
/*****************************************/
/* signon fails because we have already   */
/* signed on to this server session,      */
/* so rhost2 will be set to 2 to          */
/* indicate this, but rhost1 will         */
/* still be the MACVAR associated         */
/* with rhost.                            */
/*****************************************/
signon rhost macvar=rhost2;
rsubmit rhost wait=no;
   data a;
   x=1;
   run;
endrsubmit;
/*****************************************/
/* rhost1 is still the default and        */
/* will indicate the progress of any      */
/* subsequent RSUBMITs.                   */
/*****************************************/
%put &rhost1;
```

SIGNOFF Statement

Ends the connection between a client session and a server session.

Valid in: Client session

Syntax

SIGNOFF <*options*>;

Optional Arguments

ALL
ends all client/server connections in parallel.

If multiple server sessions are active, and _ALL_ is not specified, then the most recently signed on server session is signed off.

If you use a script for sign-on without using the URL or FTP options, then the script file will be used to perform the sign-off. For information about the URL and FTP options in the FILENAME statement, see “FILENAME” on page 201.

If the CMACVAR= option is specified in the SIGNON statement, but not in the SIGNOFF _ALL_ statement, the macro variable will be updated during the execution of SIGNOFF _ALL_.

If the CMACVAR= option is specified in the SIGNOFF _ALL_ statement, only that macro variable is updated. Any macro variables that were specified in the SIGNON statement will be ignored.

See Table 7.5 on page 148 for values that can be returned when you use the CMACVAR= option for individual task IDs when signing off.

See Table 7.6 on page 148 for values that can be returned when you use the CMACVAR= _ALL_ option when signing off.

When you run multiple and independent SAS sessions (asynchronous tasks) on the same multiprocessor machine and a SIGNOFF command is issued, the asynchronous tasks are converted to synchronous tasks and are completed before the completion of the signoff.

CMACVAR=*value*
specifies the name of the macro variable to associate with the sign-off. When CMACVAR= is specified, SAS generates a return code that provides information about the state of the sign-off. Except for this condition, the macro variable is set after the SIGNOFF command is completed.

Note: If the SIGNOFF command fails because of incorrect syntax, then the macro variable is not set.

Table 7.2 CMACVAR Macro Variable Values in SIGNOFF for Individual Task IDs

Value	Description
0	Indicates that the sign-off was successful
1	Indicates that the sign-off failed
2	Indicates that the sign-off was unnecessary

If the CMACVAR= option is specified in the SIGNOFF _ALL_ statement, only that macro variable is updated.

Table 7.3 CMACVAR Macro Variable Values in SIGNOFF with _ALL_ Option Specified

Value	Description
0	Indicates that all sign-offs were successful
1	Indicates that at least one sign-off failed
2	Indicates that the sign-offs were unnecessary

Alias MACVAR=

Interaction If SYSERR is being used and it is already set to 1012 due to a previous error in a SIGNON, RSUBMIT, or SIGNOFF statement, then it will not be reset to 0 after submitting a subsequent successful SIGNON, RSUBMIT, or SIGNOFF statement. Because SYSERR is reset only at step boundaries, you can reset its value by performing a valid DATA step or PROC step. For more information

about the SYSERR automatic macro variable, see "SYSERR" in *SAS Macro Language: Reference*.

Example "Example 5: Use CMACVAR to Test for a Successful Sign-on " on page 146.

CONNECTREMOTE=*server-ID*
server-ID
specifies the name of the server session that you want to sign off from. If only one session is active, connectremote=*server-ID* can be omitted. If multiple server sessions are active, omitting this option signs off the most recently accessed server session. You can find out which server session is current by examining the value assigned to the CONNECTREMOTE= system option.

Alias CREMOTE=, REMOTE=, PROCESS=

CSCRIPT=*fileref* | *'filespec'*
specifies the script file to be used during sign-off. CSCRIPT can be specified as a fileref or a fully qualified pathname that is enclosed in parenthesis. If multiple CSCRIPT= options are specified, the last specification takes precedence.

fileref
is the name of the reference file that is associated with the script that ends the connection. A previously executed FILENAME statement must define the fileref.

If the fileref that you define for the script is the default fileref RLINK, you can omit this specification from the SIGNOFF command.

You might use the same script to start and end a connection. If you use one script to start and end a connection, assign only one fileref.

'*filespec*'
is the name of the SAS/CONNECT script that you want to execute. If you have not defined a fileref for the script that you want to execute, use the filespec in the SIGNOFF command. The filespec can be either a fully qualified filename or the name of a file in the current working directory.

Do not specify both a fileref and a filespec.

Alias SCRIPT=

NOCSCRIPT
specifies that no SAS/CONNECT script should be used for sign-off. NOCSCRIPT is useful if you have defined the RLINK fileref but do not want to use it during sign-off. NOCSCRIPT accelerates sign-off and saves memory resources.

Alias NOSCRIPT

Details

The SIGNOFF command and the SIGNOFF statement end a connection between a client and a server session, and execute a script if you are using an access method that requires a script file. You can issue the SIGNOFF command from the command line in any client SAS windowing environment window or in a DM statement. You can also issue a SIGNOFF statement from the client session, which is especially

useful for interactive line mode sessions or non-interactive jobs. The SIGNOFF command and the SIGNOFF statement run synchronously.

Examples

Example 1: Check for Sign-off Failures

In this example, a macro variable is assigned at sign-on. Therefore, if the sign-off fails, the macro variable will be set for this server session.

```
   /* Sign-on successful, rhost1 will be  */
   /* set to 0 to indicate success, and                */
   /* macro variable rhost1 is now                     */
   /* associated with this server                      */
   /* session.                                        * /
signon rhost cmacvar=rhost1;
   /* Sign-off will fail, and rhost2                   */
   /* will be set to 1 to indicate this,               */
   /* but because it was unsuccessful,                 */
   /* rhost1 is still the default macro                */
   /* variable associated with this                    */
   /* server session.                                  */
signoff rhost cmacvar=rhost2
   cscript='noexist.scr';
```

Example 2: Simple Sign-off for a Single Session

The following FILENAME statement assigns the fileref RLINK to a SAS/CONNECT script that is named *external-file-name*:

```
filename rlink 'external-file-name';
```

Because the client is connected to only one server session, a short form of the SIGNOFF statement can be used to end the connection:

```
signoff;
```

Example 3: Sign Off from a Specific Session

If multiple server sessions are executing, you can specify the *server-ID* of the server from which to sign off.

```
signoff ahost;
```

Example 4: Sign Off from Session Using Specific Script Fileref

The following FILENAME statement assigns another fileref, which is not the default, to the SAS/CONNECT script:

```
filename endit 'external-file-name';
```

In this case, you must specify the fileref in the SIGNOFF statement because it is not the default script fileref.

```
signoff cscript=endit;
```

Example 5: Sign Off By Using a File Specification When Multiple Sessions Are Running

If you do not assign a fileref to the SAS/CONNECT script, you must specify the filespec in the SIGNOFF command.

```
signoff  all cscript='external-file-name';
```

Example 6: Sign Off without a Script

If you do not want to perform any special processing when you sign off, you can omit the script that is used for signing off.

```
signoff noscript;
```

8

RSPT Statements

Dictionary

RSPT Statement

Statements used for remote SQL pass-through.

Valid in: client session

Syntax

CONNECT TO *dbms-name* <AS *alias*> <(*dbms-argument-1=value* <*dbms-argument-2=value*>…)>;

SELECT . . . FROM CONNECTION TO *dbms-name* | *alias* (*dbms-query*);

EXECUTE (*SQL-statement*) **BY** *dbms-name* | *alias*;

DISCONNECT FROM *dbms-name* | *alias*;

CONNECT TO REMOTE <AS *alias*>
(SERVER=*serverid* <SAPW=*server-access-password*>
<DBMS=*dbms-name*>
<PT2DBPW=*passthrough-to-DBMS-password*>
<DBMSARG=(*dbms-argument-1=value* <*dbms-argument-2=value*>…)>);

SELECT . . . FROM CONNECTION TO REMOTE | *alias* (*dbms-query*);

EXECUTE (*SQL-statement*) **BY** REMOTE | *alias*;

DISCONNECT FROM REMOTE | *alias*;

Syntax Description

SERVER=*server-ID*
identifies the name of the SAS server. If the SAS/SHARE multi-user server is used, *server-ID* is the name specified for the ID= option in the PROC SERVER statement. If the SAS/CONNECT single-user server is used, *server-ID* specifies the server session. In either case, *server-ID* should be the same name that is specified in the SERVER= option in a LIBNAME statement.

SAPW=*server-access-password*
specifies the password for controlling user access to a multi-user server as specified in the UAPW= option in the PROC SERVER statement. If UAPW= is specified when the server is started, you must specify SAPW= in a CONNECT TO REMOTE statement that specifies that server.

DBMS=*dbms-name*
identifies the remote DBMS to connect to. This is the same name that you would specify in a CONNECT TO statement if you were connecting directly to the DBMS. This option is used if you want to connect to a remote DBMS instead of the remote SAS SQL processor.

PT2DBPW=*passthrough-to-DBMS-password*
specifies the password for controlling pass-through access to remote DBMS databases that are specified by using the PT2DBPW= option in the PROC SERVER statement. If PT2DBPW= is specified when the server is started, you must specify PT2DBPW= in a CONNECT TO REMOTE statement that specifies the same server and specifies DBMS=.

DBMSARG=(*dbms-argument-1=value ... <dbms-argument-n=value>*)
specifies the arguments that are required by the remote DBMS to establish the connection. These are the same arguments that you would specify in a CONNECT TO statement if you were connecting directly to the DBMS.

FROM CONNECTION TO REMOTE | *alias* (*dbms-query*);
specifies the connection to the remote SAS SQL processor or the remote DBMS as the source of data for the SELECT statement and the recipient of the *dbms-query*. For remote SAS data that is accessed through the PROC SQL view engine, *dbms-query* is any valid SELECT statement in PROC SQL. For a remote DBMS, *dbms-query* is the same SQL query that you would specify if you were connected directly to the DBMS.

EXECUTE (*SQL-statement*) BY REMOTE | *alias*;
specifies an SQL statement to be executed by the SAS SQL processor or by the remote DBMS in the server session. For remote SAS data that is accessed through the PROC SQL view engine, *SQL-statement* is any valid PROC SQL statement except SELECT. For a remote DBMS that is accessed through a single-user server in a SAS/CONNECT session, *SQL-statement* is the same SQL statement that you would specify if you were connected directly to the DBMS. For a remote DBMS, this statement might not be used if the DBMS is accessed through a remote multi-user server.

DISCONNECT FROM REMOTE | *alias*;
ends the connection to the remote DBMS or to the SAS SQL processor in the server session.

Details

Compute Services and RSPT

Remote SQL pass-through (RSPT) gives you control of where SQL processing occurs. RSPT enables you to pass SQL statements to a remote SAS SQL processor by passing them through a remote SAS server. You can also use RSPT to pass SQL statements to a remote DBMS by passing them through a remote SAS server and a Remote access engine that supports pass-through.

You can use RSPT to reduce network traffic and to shift CPU load by sending queries for remote data to a server session. (If the server is a SAS/CONNECT single-user server that you can also RSUBMIT queries to achieve the same goals.)

For example, this code contains the libref SQL that points to a server library that is accessed through a SAS/CONNECT or a SAS/SHARE server. Each row in the table EMPLOYEE must be returned to the client session in order for the summary functions AVG() and FREQ() to be applied to them.

```
select employee_title as title, avg(employee_years),
   freq(employee_id)
      from sql.employee
      group by title
      order by title;
```

However, this code contains a query that is passed through the SAS server to the SAS SQL processor, which processes each row of the table and returns only the summary rows to the client session.

```
select * from connection to remote
   (select employee_title as title,
    avg(employee_years),
    freq(employee_id)
       from sql.employee
       group by title
       order by title);
```

You can also use RSPT to join server data with client data. For example, you can specify a subquery against the DB2 data that is sent through the SAS server to the DB2 server. The rows for the divisions in the southeast region are returned to your client session, where they are joined with the corresponding rows from the local data set MyLib.Sales08.

```
libname mylib 'c:\sales';
proc sql;
   connect to remote
      (server=tso.shr1 dbms=db2
       dbmsarg=(ssid=db2p));
   select * from mylib.sales08,
      connection to remote
         (select qtr, division,
                 sales, pct
            from revenue.all08
            where region='Southeast')
      where sales08.div=division;
```

If your server is a SAS/CONNECT single-user server, you can also use RSPT to send non-query SQL statements to a remote DBMS. For example, this code sends the SQL DELETE statement through the SAS server to the remote Oracle server.

```
proc sql;
   connect to remote
      (server=sunserv dbms=oracle dbmsarg=(user=scott password=tiger);
   execute (delete from parts.inventory
      where part_bin_number='093A6')
      by remote;
```

For more information about remote SQL pass-through, see Figure 1.2 on page 8.

Examples

Example 1: RSPT Services: Query a Table in DB2

This example shows how to query a DB2 table that is located on a server by using SQL statements issued from a client session.

This code is used in a z/OS client session to connect to DB2 and query the table SYSIBM.SYSTABLES:

```
connect to db2 (ssid=db2p);

select * from connection to db2
   (select name, creator, colcount
       from sysibm.systables
       where creator='THOMPSON' or
             creator='JONES');
```

The same connection and query could be performed in a Windows client session by using RSPT by means of a z/OS server session:

```
connect to remote
   (server=rmt dbms=db2 dbmsarg=(ssid=db2p));
select * from connection to remote
   (select name, creator, colcount
       from sysibm.systables
       where creator='THOMPSON' or
             creator='JONES');
```

Using the AS alias clause in the CONNECT TO statement gives the connection name to the remote DBMS as if connected directly to it. Using this alias enables you to use queries without changing the FROM CONNECTION TO clause:

```
connect to remote as db2
   (server=rmt dbms=db2 dbmsarg=(ssid=db2p));

select * from connection to db2
    (select name, creator, colcount
        from sysibm.systables
        where creator='THOMPSON' or
              creator='JONES');
```

Example 2: RSPT Services: Subset Remote SAS Data

Four variations of the code are used to generate a PROC SQL view named Sales08, which presents sales data for fiscal year 2008. Here are the variations:

RSPT: Server Processing and Client Viewing

The data set is subsetted in the server session where summary function (SUM) is applied. Only the summary row is returned to the client session.

Processing this view is relatively fast because the data is summarized in the server session and only the resulting view is returned to the client session.

```
create view servlib.sales08 as
   select customer, sum(amount) as amount
   from sales
   where year=2008 and
      salesrep='L. Peterson'
      group by customer
      order by customer;
```

RSPT: Client Processing and Viewing

The client uses RSPT to process server data in the client session and to create the final view in the client session.

This code creates a PROC SQL view in a SAS library in the client session, which uses RSPT to access the remote SAS data from the server session:

Note: The libref ServLib can be defined for the server SAS library either in the client or the server session. In this example, a LIBNAME statement is executed in the client session to access the library that is located on the server. Alternatively, you could remotely submit a LIBNAME statement to define the library in the server session.

```
libname mylib 'C:\sales';

libname servlib '/dept/sales/revenue' server=servername;

proc sql;
connect to remote
   (server=servername);
```

```
create view mylib.sales08 as
   select * from connection to remote
      (select customer, sum(amount) as amount
      from servlib.sales
      where year=2008 and
            salesrep='L. PETERSON'
      group by customer
      order by customer);
```

RSPT: Server Processing and Viewing

The client uses RSPT to process server data in the server session and to present the final view in the server session.

In the server session, you might want to create a view that can be used by many people. By modifying the previous example to include all sales representatives, the view satisfies the needs of users who are interested in the sales that are made by more than one sales representative.

This example creates a view in the server session that summarizes the data by customer for all sales representatives:

```
libname servlib '/dept/sales/revenue'
   server=servername;

proc sql;
connect to remote
   (server=servername);

execute
   (create view servlib.cust08 as
      select customer,
      sum(amount) as amount from sales
      where year=2008
      group by customer) by remote;
```

RLS: Client Processing and Viewing

The client uses RLS to process server data in the client session and to create the final view in the client session. Using RLS, you can access the server data, and then subset and summarize the data and create the final view in the client session. The disadvantage of this method is the inefficient use of network resources to access the remote data and then to process the data in the client session.

```
libname mylib 'C:\sales';

libname servlib '/dept/sales/revenue'
   server=servername;

create view mylib.sales08 as
   select customer, sum(amount) as amount
      from servlib.sales
```

```
where year=2008 and
   salesrep='L. PETERSON'
   group by customer
   order by customer;
```

9

RSUBMIT Statements

Dictionary

RSUBMIT Statement

Marks the beginning of a block of statements that a client session submits to a server session for execution.

Valid in: client session

Syntax

RSUBMIT <*options*>;
ENDRSUBMIT <CANCEL>;
RDISPLAY <CONNECTREMOTE=> <*server-ID*;>
RGET <CONNECTREMOTE=> <*server-ID*>;
%SYSRPUT *macro-variable*=*value*;

%SYSLPUT *macro-variable*=*value* </REMOTE=*server-ID*>;
WAITFOR <_ANY_ | _ALL_> *task1 task2*… <TIMEOUT=*seconds*>;
LISTTASK <_ALL_ | *task*> ;
KILLTASK <_ALL_ | *task1task2*>…;

Optional Arguments

AUTHDOMAIN=*auth-domain* | "*auth-domain*"
specifies the name of an authentication domain, which is a metadata object that manages the credentials (user ID and password) that are associated with the specified domain. Specifying the authentication domain is a convenient way to obtain the metadata-based user credentials rather than having to explicitly supply them during server sign-on.

An administrator can define an authentication domain by using the User Manager in SAS Management Console.

Examples:

```
authdomain=DefaultAuth
authdomain="SAS/CONNECT Auth Domain"
```

Restriction Use the AUTHDOMAIN= option only when the AUTOSIGNON system option has been specified and a sign-on has not yet occurred.

Requirements The authentication domain and the associated credentials must be stored in a metadata repository, and the metadata server must be running in order to resolve the metadata object specification.

Enclose domain names that are not valid SAS names in double or single quotation marks.

Interaction If you specify AUTHDOMAIN=, do not also specify USERNAME= and PASSWORD=. Otherwise, sign-on is canceled.

See For complete details about creating and using authentication domains, see the *SAS Intelligence Platform: Security Administration Guide*.

SAS Management Console: Guide to Users and Permissions and SAS Management Console online Help

CMACVAR=*value*
specifies the name of the macro variable in which SAS stores a code indicating the state of the current RSUBMIT. When an RSUBMIT is executed, SAS checks the state of the RSUBMIT and stores a return code of 0, 1, or 2 in the specified CMACVAR variable.

Specifying CMACVAR= in an individual RSUBMIT restricts the macro variable to that RSUBMIT block. If multiple asynchronous RSUBMIT statements execute in the same server session, and each RSUBMIT contains a CMACVAR= specification, each macro variable will be restricted to its respective RSUBMIT block.

Note: If RSUBMIT fails because of incorrect syntax, then the macro variable is not set.

The CMACVAR macro variable can contain the following return code values:

Table 9.1 *CMACVAR Macro Variable Values in RSUBMIT*

Value	Description
0	The RSUBMIT is complete.
1	The RSUBMIT failed to execute.
2	The RSUBMIT is still in progress.

Alias MACVAR=

Interactions If the CMACVAR= option is not specified in the RSUBMIT statement but it is specified in the SIGNON statement, then the CMACVAR= option on the sign-on will be used.

The CMACVAR= option in the current RSUBMIT block will override the CMACVAR= that is specified at sign-on.

If SYSERR is being used and it is already set to 1012 due to a previous error in a SIGNON, RSUBMIT, or SIGNOFF statement, it will not be reset to 0 after submitting a subsequent successful SIGNON, RSUBMIT, or SIGNOFF statement. Because SYSERR is reset only at step boundaries, you can reset its value by performing a valid DATA step or PROC step.

See "CMACVAR=*value*" on page 128 in the SIGNON statement

Example "Example 3: The CMACVAR= Option with MP CONNECT" on page 59.

CONNECTPERSIST=YES | NO

specifies whether a connection persists (continues) or is automatically terminated after an RSUBMIT has completed. A connection results from a sign–on to the server session.

Here are the values for this option:

YES|Y specifies that a connection to the server session continues. A sign-off is not automatically performed after the RSUBMIT has completed. CONNECTPERSIST maintains the connection for subsequent RSUBMIT statements.

NO|N specifies that a connection to the server session terminates. A sign-off is automatically performed after the RSUBMIT has completed. Setting NO requires that you sign on to the server session again before you perform the next RSUBMIT.

Alias CPERSIST=, PERSIST=

Default YES

Interaction If the CONNECTPERSIST system option is also specified, the CONNECTPERSIST= option that is specified in the RSUBMIT statement takes precedence over the system option.

See "CONNECTPERSIST" on page 107

CONNECTREMOTE=<*server-ID*>
specifies the name of the server session that the RSUBMIT statements are executed in. If only one session is active, connectremote=*server-ID* can be omitted. If multiple server sessions are active, omitting this option causes the program statements to be run in the most recently accessed server session. The current server session is identified by the value that is assigned to the CONNECTREMOTE system option.

You can specify *server-ID* using the following formats:

computer-name.port-name
computer-name is the name of a server, and *port-name* is the name of the port that the spawner service runs on. If the computer name is longer than eight characters, assign the computer name to a SAS macro variable and use the macro variable name as the server ID.

This example assumes that the SAS Connect spawner was started with a service name and that name is being used.

Example
```
%let sashost=hrmach1.dorg.com;
   rsubmit connectremote=sashost.spawner-servicename;
```

computer-name.port-number
computer-name is the name of a server, and *port-number* is the port that the spawner service runs on.

CAUTION
Specifying computer-name.port-number for the server ID will fail under these conditions:

- when used in a WAITFOR statement that is used to wait for the completion of an asynchronous statement remote submit.

  Instead, use a one-level name, such as the *computer-with-port*
- when used in a LIBNAME statement.

  Instead, use a one-level name or a two-level name, such as *computer-name.__port-number*.

This example assumes that the user id and the password are provided during the signon.

Example
```
rsubmit connectremote=hrmach1.2267;
```

computer-with-port
computer-with-port is a macro variable that contains the name of a server and the port that the spawner service runs on, separated by one or more spaces. This specification is appropriate in cases where the *server-ID* must be specified as a one-level name.

This example assumes that the user id and the password are provided during the signon.

Example
```
%let sashost=hrmach1.dorg.com 2667;
   rsubmit connectremote=sashost;
```

computer-name.__port-number
computer-name is the name of a server and *port-number* is the port that the spawner service runs on. This format can be used to specify the *server-ID* value for the SERVER= option in a LIBNAME statement.

This example assumes that the user id and the password are provided during the signon.

Example
```
rsubmit connectremote=hrmach1.__2267;
```

Alias CREMOTE=, PROCESS=, REMOTE=

See "CONNECTREMOTE=" on page 108

CONNECTSTATUS=YES | NO
specifies whether the Transfer Status window is displayed for file transfers within the current RSUBMIT.

Here are the values for this option:

YES|Y specifies that the Transfer Status window is displayed for file transfers within the current RSUBMIT.

NO|N specifies that the Transfer Status window is not displayed for file transfers within the current RSUBMIT.

If the CONNECTSTATUS= option is omitted from the RSUBMIT statement, its value is resolved as follows:

1. If the CONNECTSTATUS= option is specified in the SIGNON statement, the value for the CONNECTSTATUS= option in the SIGNON statement is used.
2. If the CONNECTSTATUS system option is specified, the value for the CONNECTSTATUS system option is used.
3. Otherwise, the default behavior occurs. The default for a synchronous RSUBMIT is YES, which displays the Transfer Status window. The default for an asynchronous RSUBMIT is NO, which does not display the Transfer Status window.

Alias CSTATUS=, STATUS=

Default YES for synchronous RSUBMITs. NO for asynchronous RSUBMITs.

Interaction If the CONNECTSTATUS= option is omitted from the RSUBMIT statement, its value is resolved as follows:

See "Transfer Status Window" on page 92

"CONNECTSTATUS" on page 110

CONNECTWAIT=YES | NO
specifies whether RSUBMIT blocks execute synchronously or asynchronously. Synchronous RSUBMIT statements are executed sequentially. An RSUBMIT

must be completed in the server session before control is returned to the client session.

For asynchronous RSUBMIT statements, you can execute tasks in multiple server sessions in parallel. Control is returned to the client session immediately after an RSUBMIT begins execution to allow continued execution in the client session and in other server sessions.

Here are the values for this option:

YES|Y specifies that the RSUBMIT blocks execute synchronously.

NO|N specifies that the RSUBMIT blocks execute asynchronously.

If the CONNECTWAIT= option in RSUBMIT is omitted, the value for the CONNECTWAIT= option is resolved as follows:

1. If the CONNECTWAIT= option is specified in the SIGNON statement (or if the AUTOSIGNON system option has been specified because a sign-on has not yet occurred), the value for the CONNECTSTATUS= option in the SIGNON statement is used.
2. If the CONNECTWAIT system option is specified, the value for the CONNECTWAIT system option is used.
3. If the CONNECTWAIT= option is not specified in the SIGNON statement or if the CONNECTWAIT system option is not specified, the default for the CONNECTWAIT= option is used. The default is YES, which is to execute synchronously.

Alias CWAIT=, WAIT=

Default YES

Interactions

If the AUTOSIGNON system option has been specified and a sign-on has not yet occurred, any options that are specified in RSUBMIT are in effect for the entire connection. For example, if you specify CONNECTWAIT=NO in RSUBMIT and the AUTOSIGNON system has been specified, asynchronous RSUBMIT statements will be the default for the entire connection. However, the CONNECTWAIT= value can be overridden in individual RSUBMIT blocks. A connection is terminated using the SIGNOFF statement.

If CONNECTWAIT=NO is specified, you might also specify the CMACVAR= option. CMACVAR= enables you to programmatically test the status of the current asynchronous RSUBMIT to find out whether the task has completed or is still in progress.

When %SYSRPUT is executed within a synchronous RSUBMIT, the macro variable is defined in the client session as soon as it executes.

When %SYSRPUT is executed within an asynchronous RSUBMIT, the macro variable is defined in the client session when a synchronization point is encountered. To override this behavior, use the SYSRPUTSYNC system option.

If you sign on using the AUTOSIGNON system option with both CONNECTWAIT=NO and CONNECTPERSIST=NO, then an automatic sign-off will occur.

If an asynchronous SIGNON is followed by an asynchronous RSUBMIT, then the RSUBMIT statement is blocked until the asynchronous SIGNON is completed.

See "SYSRPUTSYNC" on page 120

"Synchronization Points" on page 193

"CONNECTWAIT" on page 111

"Example 5: MP CONNECT and the WAITFOR Statement" on page 63

CSCRIPT=*file-specification*

specifies the script file to use in an RSUBMIT when the AUTOSIGNON system option has been specified and a sign-on has not yet occurred.

file-specification

specifies the location of the script file.

Here are the values for *file-specification*:

"*filename*"

is the physical location of the script file in the current working directory. Enclose the filename in double or single quotation marks.

fileref

is a SAS name that is associated with the physical location of the script file. A previously executed FILENAME statement must define the fileref.

If the fileref that you define for the script is the default fileref RLINK, you can omit this specification from RSUBMIT.

"*fully-qualified-filename*"

is the full path to the script file. Enclose the fully qualified filename in double or single quotation marks.

"*SASSCRIPT-specification*"

is the physical location of the script file in the directory that is specified by the SASSCRIPT system option.

Alias SCRIPT=

Restriction Use the CSCRIPT= option only when the AUTOSIGNON system option has been specified and a sign-on has not yet occurred.

Interactions If multiple CSCRIPT= options are specified, the last specification takes precedence.

When you use the CSCRIPT= option, do not also use the NOCSCRIPT option. If you use NOCSCRIPT and CSCRIPT=, sign-on is canceled.

See " NOCSCRIPT" on page 171

"AUTOSIGNON" on page 99

FILENAME statement in *SAS DATA Step Statements: Reference* and the companion that is appropriate for your operating environment.

CSYSRPUTSYNC=YES | NO

specifies whether to synchronize the client session's macro variables when the client session receives results from the server session or when a synchronization point is encountered. Macro variables are updated in the client session using the %SYSRPUT macro in an asynchronous RSUBMIT.

Note: The %SYSRPUT macro is executed in the server session.

Here are the values for this option:

YES | Y — specifies that the client session's macro variables will be updated when the client session receives the results of the server session's execution of the %SYSRPUT macro. The results are delivered in the form of a packet. Specifying YES does not mean that the client's macro variables will be updated immediately after the server session's execution of the %SYSRPUT macro variable. YES means that the client's macro variables will be updated when the client receives the packet from the server session. Therefore, the exact time at which the client session's macro variables are updated will depend on the availability of the client session to receive the packet from the server session. If the client session is busy, the server session must wait until the client session is ready to receive the packet.

NO | N — specifies that the client session's macro variables will be updated when a synchronization point is encountered. This is the default.

Alias SYSRPUTSYNC=

Default NO

Interactions If the SYSRPUTSYNC system option is specified, the CSYSRPUTSYNC= option in RSUBMIT takes precedence over the system option.

If the SYSRPUTSYNC system option is specified and the CSYSRPUTSYNC= option in RSUBMIT is not specified, the system option will apply to the RSUBMIT statement.

Changing the value assigned to the SYRPUTSYNC= option between consecutive asynchronous RSUBMIT statements causes unpredictable results. You are advised not to change the value between asynchronous RSUBMIT statements.

See “Synchronization Points” on page 193

SASCSCRIPT system option on page 120

“FILENAME” in *SAS Global Statements: Reference* for an example of how to prevent SYSRPUTSYNC= option overrides.

INHERITLIB=(*client-libref1 <=server-libref1> ... client-librefn <=server-librefn>*)
enables libraries that are defined in the client session to be inherited by the server session for Read and Write access. As an option, each client libref can be associated with a libref that is named differently in the server session. If the server libref is omitted, the client libref name is used in the server session. A space is used to separate each libref pair in a series, which is enclosed in parenthesis.

Note: Because the SAS Work library cannot be reassigned in any SAS session, you cannot reassign the SAS Work library in the server session either.

Restriction The INHERITLIB= option is not supported in either the SIGNON or the RSUBMIT statements to start a secondary (nested) SAS/CONNECT session in a remote SAS/CONNECT server session. If you use the option this way, the secondary session will continue, but the option will be ignored and a WARNING is sent to the SAS log.

Interactions If you use the INHERITLIB= option and the SASCMD= option when signing on to a server session, then the server session attempts to access the client library directly rather than to inherit access to the library via the client session. If the client session and the server session attempt to access the same file simultaneously, only one session is granted exclusive access to the file. The other session's access to the file is denied.

SAS/CONNECT does not support concurrent multi-user access to the same file. This functionality is supported by SAS/SHARE.

See SASCMD= on page 174

SAS/SHARE User's Guide

Example This example shows that the libref named Local in the client session is inherited for use in the server session.

```
rsubmit job1 inheritlib=(local work=remote);
   libname local list;
   libname remote list;
   data local.a;
   x=1;
   run;
endrsubmit;
```

LOG=KEEP | PURGE | *file-specification*< NEW >
OUTPUT=KEEP | PURGE | *file-specification*< NEW >
directs the SAS log or the SAS output that is generated by the current server session to the backing store or to the specified file. A *backing store* is a SAS utility file that is written to the client SAS Work directory.

Here are the values for these options:

KEEP
spools log or output lines, as applicable, to the backing store or to the computer on which the client session is running. The log or output lines can be retrieved using the RGET, RDISPLAY, RSUBMIT CONNECTWAIT=YES, or SIGNOFF statements. This is the default.

PURGE
deletes all the log or output lines that are generated by the current server session. PURGE is used to save disk resources. If you do not need the data, you can use PURGE to remove large volumes of log or output data that are written to the backing store.

***file-specification* < NEW >**
specifies a file as the destination for the log or output lines. The file is opened for output at the beginning of the asynchronous RSUBMIT and is closed at the end of the asynchronous RSUBMIT. After the current RSUBMIT has completed, subsequent RSUBMIT log or output lines can be appended to the preceding RSUBMIT destination file using the LOG= or OUTPUT= options. If you specify the same filename for multiple RSUBMIT statements and you do not specify the NEW or MOD options, then the log data will be appended to the current file by default.

Note: Directing output to the same file for multiple concurrent asynchronous RSUBMIT statements is not recommended

Here are the values for this option:

"filename "
is the physical location of the SAS log file or the SAS output file. Enclose the filename in double or single quotation marks.

fileref
is a SAS name that is associated with the physical location of the SAS log file or the SAS output file.

Note Use the MOD option in the FILENAME statement to open the referenced file for an append. The MOD option is an external I/O statement option.

NEW
specifies that the file will be opened for new log output. For example, if the file already exists from previous RSUBMIT sessions, it is deleted and re-created rather than appended to the current output log file.

The NEW option takes precedence over any options specified in the FILENAME statement. For example, the MOD option in the FILENAME statement in SAS causes output to be appended to an existing file. If you specify the MOD in the FILENAME statement with the NEW option in the RSUBMIT statement simultaneously, then the NEW option will be honored and the specified file will be opened for new output rather than appended.

```
filename myLog "d:\reports";
SIGNON session1 sascmd="!sascmd -nosyntaxcheck -noterminal
                                -noconnectwait";
   rsubmit wait=no log=myLog new;
     data a;
     t=1;
   run;
   endrsubmit;
signoff session1;
```

Default KEEP

Restriction Use the LOG= and the OUTPUT= options only when executing an asynchronous RSUBMIT. Otherwise, a WARNING will be displayed in the log and the options will be ignored.

Interactions If you use both the asynchronous RSUBMIT and the PROC PRINTTO statements at the same time, the statements will execute simultaneously making it impossible to predict which operation will complete first. If the PROC PRINTTO executes first so that data from the server session can be written to the specified PROC PRINTTO file, then the LOG= (or the OUTPUT=) option in the SIGNON statement is ignored, and no data is written to the specified file.

However, because the asynchronous RSUBMIT and the PROC PRINTTO statements execute simultaneously, predicting which operation will complete first is impossible. The timing of the completions of these operations determines whether the results are written to the SIGNON log or to the PROC PRINTTO log.

If you direct the log or output lines to a file and then use RGET or RDISPLAY to retrieve the contents of an empty backing store, this message is displayed:

```
WARNING: The LOG option was used to file
log lines for the current RSUBMIT.
There are no log lines for RGET to process.
```

Note Do not simultaneously use an asynchronous RSUBMIT and the PROC PRINTTO statement to redirect output. Results are unpredictable when you use a LOG= or an OUTPUT= option to redirect output in an asynchronous RSUBMIT and then use the PROC PRINTTO statement in the client session.

See CONNECTWAIT= option on page 165

"Example 8: Force Macro Variables to Be Defined When %SYSRPUT Executes" on page 66

NOCSCRIPT

specifies that no script file should be used for sign-on. NOCSCRIPT accelerates sign-on and conserves memory resources.

Alias NOSCRIPT

Restriction Use the NOCSCRIPT option only when the AUTOSIGNON system option has been specified and a sign-on has not yet occurred.

Interaction When you use NOCSCRIPT, do not also use SASCMD=, SERVER=, or CSCRIPT=. If you use NOCSCRIPT with SASCMD=, NOCSCRIPT is ignored. If you use NOCSCRIPT with SERVER= or CSCRIPT=, sign-on is canceled.

```
WARNING: The LOG option was used to
file log lines for the current RSUBMIT.There are no
log lines for RGET to process.
```

See "AUTOSIGNON" on page 99

"CSCRIPT=*file-specification*" on page 167

NOTIFY=YES | NO | "*e-mail-address*"
specifies whether to notify the user that an asynchronous RSUBMIT has completed. The notification can be in the form of a message window or an email message. The NOTIFY option is enabled only at sign-on and remains in effect for the duration of the server session.

Here are the values for this option:

YES|Y	enables notification via a message window. Here is the format of the default message: **`Asynchronous task TASK1 has completed.`**TASK1 is the server ID. The message window does not interfere with any other task executions in progress. To acknowledge the message and to close the window, click **OK**.
NO|N	disables notification. This is the default.
"*e-mail-address*"	enables notification via an email message, and specifies the email address of the recipient for the notification. Email addresses are limited to a maximum of 256 characters. Enclose the email address in double or single quotation marks. The message includes information about the total time that was used for the asynchronous RSUBMIT. If the LOG= and OUTPUT= options are also specified in an asynchronous RSUBMIT statement, the email message identifies the locations of the log file and output file.

Here is an example of enabling notification for an asynchronous RSUBMIT:

```
options autosignon sascmd="!sascmd";
rsubmit process1 wait=no notify=yes;
   %put should get notification window;
endrsubmit;
```

To disable notification, you must sign off from the server session and then sign on to the server session again. When you sign on again, either omit the NOTIFY= option or specify NOTIFY=NO in the RSUBMIT statement.

Here is an example of disabling notification for the next asynchronous RSUBMIT:

```
signoff process1;
options autosignon sascmd="!sascmd";
rsubmit process1 wait=no notify=no;
   code-to-be-executed-in-server-session
endrsubmit;
```

Default NO

Restrictions Notification occurs only for asynchronous RSUBMIT statements.

If NOTIFY=YES or NOTIFY="*e-mail-address*" is specified in a synchronous RSUBMIT block, notification fails. Notification is valid only for an asynchronous RSUBMIT.

Use the NOTIFY= option in RSUBMIT only when the AUTOSIGNON system option has been specified (because a sign-on has not yet occurred).

If NOTIFY= is specified in RSUBMIT when the AUTOSIGNON system option has been specified, but a sign-on has previously occurred, NOTIFY= has no effect.

Interactions When you specify the NOTIFY="*e-mail-address*" option, you can also specify the SUBJECT="*subject-title*" option.

If NOTIFY=YES and the NOTERMINAL system option has been specified, the request for notification is ignored. This message is displayed:

```
WARNING: The NOTIFY option is valid
only if a TERMINAL is attached
to this SAS session. Option will be ignored.
```

However, notification can be directed to an email address, regardless of whether the TERMINAL or NOTERMINAL system option has been specified.

If NOTIFY="*e-mail address*" is specified, but the email message cannot be sent, notification will occur in the form of a message window, which is the action that occurs when NOTIFY=YES. This behavior assumes that the NOTERMINAL system option has not been specified.

If NOTIFY="*e-mail address*" is specified, the SAS system and the operating environment that the SAS system runs under must be configured to support email. Without appropriate configuration, your attempt to specify notification via email might fail. Contact your system administrator for details.

Notification fails if NOTIFY=YES or NOTIFY="*e-mail address*" and you specify statements or commands (such as RGET or SIGNOFF) during the asynchronous RSUBMIT that change execution from asynchronous to synchronous mode.

This message is displayed when the NOTIFY= option is specified in the RSUBMIT statement:

```
WARNING: The NOTIFY option is applied
only during SIGNON, but remains in effect for the
entire connection until SIGNOFF.
```

This message is also displayed for an RSUBMIT for which an automatic sign-on has occurred.

See CONNECTWAIT=NO option on page 165

AUTOSIGNON System Option on page 99

LOG= and OUTPUT= options on page 133

“SUBJECT="*subject-title*"” on page 177

EMAILHOST, EMAILPORT, and EMAILSY system options in *SAS System Options: Reference*

PASSWORD=*password* | "*encoded-password*" | _PROMPT_
specifies the password to use in order to sign on to a server session. The operating environment that the server session runs under can affect password naming conventions. For details about password-naming conventions according to operating environment, see UNIX on page 339, z/OS on page 361, and Windows on page 383.

Here are the values for this option:

password
The value for this option is replaced by Xs in the log. To protect this password, you should use the security software at your site to limit access to the SAS program statements that create the server.

See For details about valid passwords and user IDs, see "User ID and Password Naming Conventions" on page 144.

"*encoded-password*"
is an encoded version of a password. Using encoded passwords promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password as input to the PWENCODE procedure. For information about using PROC PWENCODE to create an encoded password, see the PASSWORD= on page 137 option in the SIGNON statement.

Here is an example of code for obtaining an encoded password:

```
 proc PWENCODE in="srvmach";
run;
{sas001}c2Vydm1hY2g=
```

The clear-text password `srvmach` is specified in the PROC PWENCODE statement. The output is generated in the form `{key}encoded-password`. `sas001` is the key, which is used to decode the encoded password to its clear-text form when the password is needed.

Note: The encoded password is case sensitive. Use the entire generated output string, including the key.

Use the output from the PROC PWENCODE statement as the value for *encoded-password* in the appropriate statement.

PROMPT specifies that SAS prompt the user for a valid password. This value enforces security.

Alias PASSWD=, PASS=, PWD=, PW=

Restriction Use the PASSWORD= option only when the AUTOSIGNON system option has been specified (because a sign-on has not yet occurred).

See "AUTOSIGNON" on page 99

SASCMD="*SAS-command*" | sascmd" | sascmdv" | "*host-command-file*"
signs on to the server session on the same symmetric multiprocessing (SMP) computer that the client session is running on. This option is most useful when client and server sessions run on SMP hardware.

"*SAS command*"
- For UNIX and Windows: specifies the SAS command that is used to sign on to a server session.

  Here is a typical example:

```
sascmd="sas"
```

As another example, commands that contain spaces must be enclosed in double quotation marks.

```
sascmd='"c:\Program Files\SAS\SAS System\9.2\sas.exe"';
```

- For z/OS: specifies a colon that is followed by any SAS invocation options.

```
sascmd=":ls=256"
```

Here is an example:

!sascmd
For UNIX and Windows, signs on to a server session by using the same command that was used to start the client session.

!sascmdv
For UNIX and Windows, signs on to a server session by using the same command that was used to start the client session. The SAS invocation is written to the SAS log.

"host-command-file"
To execute additional commands before SAS is invoked, you can write a command file that is specific to your operating environment. Here are the file extensions according to operating environment: Windows filenames use the `.bat` and `.cmd` extensions. UNIX extensions include `.sh`, `.csh`, and `.ksh`. The SASCMD= option does not support z/OS command files.

The TCP/IP access method adds options, such as -DMR, to the server session's SAS command.

For Windows, the TCP/IP access method also appends these options:

- -COMAMID TCP
- -ICON
- -NOSPLASH
- -NOTERMINAL

NODETACH causes a sign-on to occur in a subprocess of the parent's process, which can use excessive resources. If NODETACH is specified, try setting the DETACH system option, which causes sign-ons to occur as detached processes rather than as subprocesses.

Restriction For z/OS, a command file cannot be used. Therefore, use a semicolon followed by options for the server's SAS invocation.

Requirement SAS commands that contain spaces must be enclosed in double or single quotation marks.

Interactions If the SASCMD= system option is already specified, the SASCMD= option that is specified in the RSUBMIT statement block takes precedence over the system option.

When you use SASCMD=, do not also use NOCSCRIPT. Otherwise, NOCSCRIPT is ignored.

See “SASCMD=” on page 113

SYNTAXCHECK= and NOSYNTAXCHECK= system options in *SAS System Options: Reference*

ICON, NOSPLASH, and NOTERMINAL system options in *SAS Companion for Windows*

"COMAMID=" on page 101

" NOCSCRIPT" on page 171

SERVER="*SAS-application-server*"
specifies the name of a SAS Application Server that contains a SAS/CONNECT server component in its grouping. The SAS Application Server has been defined in the SAS Metadata Repository using SAS Management Console. The SAS Application Server is configured using a set of system resources, including a SAS/CONNECT server component and properties that start a SAS/CONNECT server session. The server properties are equivalent to the options that can be specified in the SIGNON statement.

When you use the SERVER= option, certain system resources must be configured before you can access a SAS Metadata Server. For details, see "Metadata Server-based Sign-ons" on page 18.

"*SAS-application-server*"
specifies a SAS Application Server that contains a SAS/CONNECT server component, which has been defined in a SAS Metadata Repository.

Requirements Enclose the name of the SAS Application Server in double or single quotation marks.

If the specified SAS Application Server does not contain a SAS/CONNECT server component, the server sign-on fails.

Interactions SERVER= is used in an RSUBMIT when an automatic sign-on is in effect via the AUTOSIGNON system option rather than when an explicit sign-on is specified via the SIGNON statement.

When you use SERVER=, do not also use these RSUBMIT options: NOCSCRIPT, NOTIFY=, PASSWORD=, REMOTE=, SASCMD=, SCRIPT=, SIGNONWAIT=, or USERNAME=. Here is an example of a warning:

```
WARNING: NOTIFY= and SERVER= are mutually exclusive.
Please choose only one of them.
```

If any of these options is also specified, the entire RSUBMIT code block will be ignored. Although the AUTOSIGNON system option might be in effect, a server sign-on will fail.

When you use SERVER=, you can also specify any of these options in RSUBMIT: CMACVAR=, CONNECTPERSIST=, CSTATUS=. CWAIT=, INHERITLIB=, LOG=, OUTPUT=, OUTPUT=, or SYSRPUTSYNC=. These options, when specified in an RSUBMIT, take precedence over the equivalent options in the SAS/CONNECT component of the SAS Application Server.

If you use NOCSCRIPT and SERVER=, sign-on is canceled.

The CONNECTPERSIST= and SYSRPUTSYNC= options are available only in the RSUBMIT statement. They cannot be specified as sign-on properties for the SAS/CONNECT component that is contained in the SAS Application Server.

See "SERVERV="*SAS-application-server*" | _ALL_" on page 140 in SIGNON

"AUTOSIGNON" on page 99

SAS Management Console: Guide to Users and Permissions and SAS Management Console online Help

SIGNONWAIT=YES | NO

specifies whether a sign-on to a server session is to be executed synchronously or asynchronously. You can sign on using the SIGNON statement or the AUTOSIGNON system option.

Here are the values for this option:

YES | Y — specifies a synchronous sign-on. A synchronous sign-on causes the client session to wait until the sign-on to a server session has completed before control is returned to the client session for continued execution. YES is the default.

NO | N — specifies an asynchronous sign-on. An asynchronous sign-on to a server session begins execution and control is returned to the client session immediately for continued execution. Asynchronous sign-on allows multiple tasks (including other sign-ons) to be executed in parallel. Asynchronous sign-ons reduce the total amount of time that would be used to execute individual sign-ons to multiple server sessions. Using the saved time, the client session can execute more RSUBMIT statements.

Default YES

Interactions If the SIGNONWAIT system option is also specified, the SIGNONWAIT= option takes precedence over the system option.

If SIGNONWAIT is specified as a system option and the SIGNONWAIT= option is not specified, the system option will apply to the RSUBMIT statement.

If SIGNONWAIT=NO is specified, the USERID= and PASSWORD= options cannot be set to _PROMPT_.

See "SIGNONWAIT" on page 118

"AUTOSIGNON" on page 99

"SIGNON" on page 127

SUBJECT="*subject-title*"

specifies the subject title for the email notification message that is sent after an asynchronous RSUBMIT completes. A subject title is limited to a maximum of 256 characters.

Here is an example of specifying a subject using email notification:

```
options remote=myhost sascmd="!sascmd";
signon notify="joe.smith@apex.com";
rsubmit wait=no subject="First task completed on &SYSHOSTNAME";
   code-to-be-executed
endrsubmit;
```

Restriction Use the SUBJECT= option only when the NOTIFY="*e-mail-address*" option is in effect.

Interaction If the SUBJECT= option is specified at sign-on, but not specified in the RSUBMIT statement, then the subject title that is specified at sign-on is used in the email message for the RSUBMIT. If no SUBJECT= is specified, the default subject title is used:

```
SAS/CONNECT task TASK1 has completed.
```

TASK1 is the server ID.

See " NOTIFY=YES | NO | "*e-mail-address*"" on page 172

"NOTIFY Script" on page 298

SAS system options that support email configuration: and "EMAILPORT" in *SAS System Options: Reference* in *SAS System Options: Reference*

USERNAME=*user-ID* | _PROMPT_
specifies the user ID to be used when connecting to a server session.

user-ID
specifies the name to be used when using the RSUBMIT statement to submit code to a remote server session.For details about a valid user ID, see "User ID and Password Naming Conventions" on page 144.

PROMPT
specifies that SAS prompt the user for a valid user ID. This value enforces security.

Alias USERID=, USER=, UID=

Restriction Use the USERNAME= option only when the AUTOSIGNON system option has been specified (because a sign-on has not yet occurred).

See "AUTOSIGNON" on page 99

For details about a valid user ID, see "User ID and Password Naming Conventions" on page 144.

Details

Table of Procedure Tasks and Examples

Task	Statement
Mark the end of a block of statements that a client session submits to a server session for execution	"ENDRSUBMIT" on page 181
Create a Log window to display the lines from the log. Create an Output window to list the output generated	"RDISPLAY" on page 182

from the execution of the statement within an asynchronous RSUBMIT block	
Retrieve the log and output that are created by an asynchronous RSUBMIT and merge them into the Log and Output windows of the client session	“RGET” on page 183
Assign a value from the server session to a macro variable in the client session	“%SYSRPUT” on page 191
Create a macro variable in the server session	“%SYSLPUT” on page 184
Cause the client session to wait for the completion of one or more tasks (asynchronous RSUBMITs) that are in process	“WAITFOR” on page 195
List all active connections or tasks and identify the execution status of each connection or task	“LISTTASK” on page 197
For an asynchronous task, force one or more active tasks or server sessions to terminate immediately	“KILLTASK” on page 198

Difference between SUBMIT and RSUBMIT

The RSUBMIT command and statement cause SAS programming statements that are entered in a client session to run in a server session. The difference between the RSUBMIT and the SUBMIT commands is the location of program execution (client session or server session). Although RSUBMIT executes tasks in a server session, results and output are delivered to the client session as if they were executed in the client session.

Difference between the RSUBMIT Statement and Command

The primary difference between the RSUBMIT command and the statement is that the command can be used only from a windowing environment session or in the DM statement. The RSUBMIT statement is used in a client session.

You can use the RSUBMIT command in these environments:

- the command line of the Program Editor window in a client session.
- a DM statement, which uses commands as if they were issued from a command line in a windowing environment.
- *Windows only:* the KEYS window in which you assign the RSUBMIT command to a key. For details, see the *SAS Companion for Windows*.

Difference between Synchronous and Asynchronous RSUBMITs

An RSUBMIT is executed either synchronously or asynchronously.

synchronous
: Client session control is not returned until the RSUBMIT has completed. Synchronous execution is the default execution mode.

asynchronous
: Client session control is returned immediately after an RSUBMIT is sent to a server session. Program execution can occur in a client session and in one or more server sessions in parallel.

A synchronous RSUBMIT displays results and output in the client session. If the RSUBMIT is asynchronous, you can use the RGET and RDISPLAY commands and statements and the LOG= and OUTPUT= options to retrieve and view the results.

Execute Statements in the RSUBMIT Block

The RSUBMIT command can be used to execute most types of SAS programs in the server session, except windowing procedures (such as SAS/FSP or SAS/AF procedures).

The RSUBMIT statement can be used to run SAS/CONNECT from the SAS windowing environment, an interactive line mode session, or a batch job. The RSUBMIT and the ENDRSUBMIT statements together constitute the *RSUBMIT block*. This RSUBMIT block enables you to separate the server-session statements from the client-session statements when both are used in the same program. The statements that are enclosed in the RSUBMIT block are executed in the server session. All the other statements are executed in the client session when you run the program.

The following template can be used to build a file that includes statements for both the client and the server sessions in the same program:

```
statements for client session
rsubmit;
   statements for server session
endrsubmit;
  statements for client session
```

Note: The DOWNLOAD and the UPLOAD procedures must be executed by using the RSUBMIT command or the RSUBMIT statement. You cannot execute UPLOAD and DOWNLOAD by using the SUBMIT command.

RSUBMIT and ENDRSUBMIT Parsing

When SAS encounters an RSUBMIT statement, it sends the SAS statements in the RSUBMIT block to SAS/CONNECT. SAS/CONNECT continues parsing the statements until it encounters the semicolon that follows the ENDRSUBMIT statement.

The SAS statements within an RSUBMIT block can contain the start of a quoted string. A second RSUBMIT block can contain the end of the quoted string. Here is

an example of two RSUBMIT blocks in which a quoted string starts in the first RSUBMIT block and ends in the second RSUBMIT block:

```
rsubmit;
data _null_;
newmacro='mend;
endrsubmit;
rsubmit;
endrsubmi' || 't; ' ;
put newmacro;
run;
endrsubmit;
```

If the preceding statements were changed to have `"newmacro='mend; endrsubmit;';"` (instead of it being broken between the two RSUBMIT blocks), parsing of the RSUBMIT block would end with `"endrsubmit;"` . RSUBMIT block processing ends after the ENDRSUBMIT statement. The second quotation mark is processed in the client SAS session, so another quotation mark will need to be entered before SAS reports an error. Here is an excerpt of the error message:

```
newmacro = 'mend; endrsubmit;'
                                -
ERROR : Statement is not valid or it is used out of proper order.
```

Avoid including the ENDRSUBMIT statement in a quoted string.

ENDRSUBMIT Statement

Marks the end of a block of statements that a client session submits to a server session for execution.

Valid in: client session

Syntax

ENDRSUBMIT <CANCEL>;

Syntax Description

CANCEL
This option is useful in an interactive line mode session if you see an error in a previously entered statement, and you want to cancel the step.

Details

The ENDRSUBMIT statement signals the end of a block of statements that begins with either of the following lines of code:

```
dm 'rsubmit';
```

or

```
rsubmit;
```

The server session executes the statements between either of these statements and the ENDRSUBMIT statement.

Note: Do not use the ENDRSUBMIT statement when using the RSUBMIT command. Use it only when you use the RSUBMIT statement or the DM RSUBMIT statement.

The ENDRSUBMIT statement can be used in any type of client session: a SAS windowing environment, an interactive line mode session, or a batch job. The RSUBMIT and ENDRSUBMIT statements enable you to include in the same file statements that are executed in the client session and statements that are executed in the server session. The statements to be executed in the server session are enclosed between the RSUBMIT and ENDRSUBMIT statements.

All of the other statements in the program are executed in the client session when you run the program. Here is a template for the arrangement of statements for the server and client sessions in the same program:

```
statements for client session
rsubmit;
   statements for server session
endrsubmit;
more statements for client session
```

Note: Do not put a comment between the ENDRSUBMIT statement and the semicolon. Doing so will cause an error message to be displayed in the SAS Log and can cause unexpected results in your output.

RDISPLAY Statement

Creates a Log window to display the lines from the log and an Output window to list the output generated from the execution of the statements within an asynchronous RSUBMIT block.

Valid in: client session

Syntax

RDISPLAY <CONNECTREMOTE=*server-ID*>;

Syntax Description

CONNECTREMOTE=*server-IDserver-ID*
specifies the name of the server session that the asynchronous RSUBMIT is executing in or has executed in. If only one session is active, you can omit *server-ID*. If multiple server sessions are active and you omit this option, the spooled log and output statements from the most recently accessed server session are displayed.

Alias CREMOTE=, PROCESS=, REMOTE=

Details

The RDISPLAY command and the RDISPLAY statement create a Log window to display the spooled log and an Output window to display the output that is generated by an asynchronous RSUBMIT.

When an asynchronous RSUBMIT executes, the log and the output are not merged into the client Log and Output windows. Instead, they are spooled until they are retrieved at a later time. RDISPLAY enables you to view the spooled log and output statements that are created by the asynchronous RSUBMIT. The RGET command and the RGET statement must be used to merge the spooled statements into the client Log and Output windows.

The primary difference between the RDISPLAY command and the RDISPLAY statement is that the command can be used only from a windowing environment session or within a DM statement. The RDISPLAY statement is used in a client session.

RGET Statement

Retrieves the log and output that are created by an asynchronous RSUBMIT and merges them into the Log and Output windows of the client session.

Valid in: client session

Syntax

RGET <<CONNECTREMOTE=> *server-ID*>;

Syntax Description

CONNECTREMOTE=*server-IDserver-ID*
specifies the name of the server session that generated the spooled log and output to be retrieved. If only one session is active, *server-ID* can be omitted. If multiple server sessions are active and the option is omitted, the spooled log and output statements from the most recently accessed server session are retrieved and merged into the client Log and Output windows. You can find out which server session is the current session by examining the value that is assigned to the CONNECTREMOTE system option.

Alias CREMOTE=, PROCESS=, REMOTE=

See “CONNECTREMOTE=” on page 108

Details

The RGET command and the RGET statement cause all the spooled log and output from the execution of an asynchronous RSUBMIT to be merged into the client Log and Output windows. When an asynchronous RSUBMIT executes, the log and output are not merged into the client Log and Output windows immediately. Instead, the log and output are spooled and retrieved later.

If the RGET command or RGET statement is executed while the asynchronous RSUBMIT is still in progress, all currently spooled log and output statements are retrieved and merged into client Log and Output windows. The RSUBMIT continues execution as if it were submitted synchronously. Control is returned to the client session after the RSUBMIT has completed.

If you do not want RSUBMIT to become synchronous, but you want to check its progress, use the CMACVAR= option in the RSUBMIT or the SIGNON statement. CMACVAR= enables you to monitor the progress of an asynchronous RSUBMIT without causing it to execute synchronously.

Note: For an overview about monitoring SAS tasks, see "Monitor MP CONNECT Tasks" on page 38.

Note: For asynchronous RSUBMIT statements, the SAS system option _LAST_, which is used to find out the name of the most recently created data set, is not updated. Also, if RGET is used to change RSUBMIT execution from asynchronous to synchronous, the system option _LAST_ is not updated. For more information about _LAST_, see *SAS System Options: Reference*.

%SYSLPUT Statement

Creates a single macro variable in the server session or copies a specified group of macro variables to the server session.

Valid in: client session

Syntax

Form 1: **%SYSLPUT** *macro-variable*=*value* </REMOTE=*server-ID*>;

Form 2: **%SYSLPUT** _ALL_ | _AUTOMATIC_ | _GLOBAL_ | _LOCAL_ | _USER_ </LIKE='*character-string*'><REMOTE=*server-ID*>;

Syntax Description

ALL
copies all user-generated and automatic macro variables to the server session.

AUTOMATIC
copies all automatic macro variables to the server session. The automatic variables copied depend on the SAS products installed at your site and on your operating system. The scope is identified as AUTOMATIC.

GLOBAL
copies all user-generated global macro variables to the server session. The scope is identified as GLOBAL.

/LIKE=<*'character-string'*>
Specifies a subset of macro variables whose names match a user-specified character sequence, or pattern. Only this identified group of variables with names matching the pattern will be copied to the server session.

Note: The LIKE= option is not case sensitive.

'character-string'
Specifies the sequence of characters, or pattern, to be used as the criteria for determining which macro variables are to be copied to the server session. Character patterns can consist of the following:

- any sequence of characters, A-Z
- any sequence of digits, 0-9
- a single wildcard character in the form of an asterisk (*)

The wildcard character (*) cannot be embedded or used more than once in the character string. The examples below illustrate how the LIKE= option works with the wildcard character. For these examples, assume that the following macro variables are defined in the client session: *rc1, rc2, unixHost,* and *winHost*:

`like='rc*';`	Wildcard at the end: returns *rc1* and *rc2.*
`like='*Host';`	Wildcard at the beginning: returns *unixHost* and *winHost.*
`like='*host';`	Wildcard at the beginning and lowercase "h" in name: returns *unixHost* and *winHost.*
`like='r*c';`	Wildcard in the middle: is not valid and returns a syntax error.
`like='*rc*';`	More than one wildcard (at beginning and end): is not valid and returns a syntax error.
`like='rc';`	Wildcard not specified: returns nothing (no match)
`like=' ';`	Wildcard not specified and *'character-string'* is empty:

returns nothing (no macro variables are copied)

Restrictions The wildcard (*) cannot be embedded in the character-string.

The wildcard (*) can be specified only once in the character-string.

Requirement The wildcard (*) must be used at either the beginning or the end of the character-string.

Interaction The /REMOTE= and /LIKE= options are independent of each other and can be specified on the same %SYSLPUT statement, regardless of order.

Notes Macro variables in the same server session are over-written each time they are submitted.

Read-Only system options in the remote server are not over written.

Tip To copy all macro variables to the server session without specifying LIKE= , use the _ALL_ special word in the %SYSLPUT statement.

LOCAL
copies all user-generated local macro variables to the server session. The scope is the name of the currently executing macro.

macro-variable
specifies the name of a macro variable to be created in the server session.

value
specifies the macro variable reference, a macro invocation, or the character value to be assigned to the server *macro-variable*. The character value should not contain nested quotation marks.

Requirement Values containing special characters, such as the forward slash (/) or single quotation mark (‘), must be masked using the %BQUOTE function so that the macro processor correctly interprets the special character as part of the text and not as an element of the macro language. See “Example 3: Mask Character Values with %BQUOTE (Form 1)” on page 189 for an example of how to use the %BQUOTE function. For more information about Macro Quoting in general, see “Macro Quoting” in *SAS Macro Language: Reference*.

/REMOTE=*server-ID*
specifies the name of the server session that the macro variable will be created in. If only one server session is active, the *server-ID* can be omitted. If multiple server sessions are active, omitting this option causes the macro to be created in the most recently accessed server session. You can find out which server session is currently active by examining the value that is assigned to the CONNECTREMOTE system option.

Interactions The /REMOTE= option that is specified in the %SYSLPUT macro statement overrides the CONNECTREMOTE= system option.

The /REMOTE= and /LIKE= options are independent of each other and can be specified on the same %SYSLPUT statement, regardless of order.

See “CONNECTREMOTE=” on page 108

USER
copies all user-generated global and local macro variables to the server session. The scope is identified either as GLOBAL, or as the name of the macro in which the macro variable is defined.

Details

%SYSLPUT Macro Statement

The %SYSLPUT statement is a macro statement used in SAS/CONNECT that enables you to do the following:

- create a new macro variable in the server session and assign it a value from the client session (form 1).
- copy a specified group of existing macro variables and their values from the client to the server session (form 2).

Note: Unlike the %SYSRPUT statement that is submitted within the RSUBMIT block of code and processed in the server session, the %SYSLPUT statement is submitted outside the RSUBMIT code block and processed in the client session.

Create a Single Macro Variable to Be Used in the Server Session (Form 1)

The %SYSLPUT statement is a macro statement that is submitted in the client session to create and assign a value to a macro variable in the server session.

If you are signed on to multiple server sessions, %SYSLPUT submits the macro assignment statement to the most recently used server session. If you are signed on to only one server session, %SYSLPUT submits the macro assignment statement to that server session. If you are not signed on to any session, an error condition results.

When you use asynchronous SIGNONs and RSUBMITs, the implicit last-used session ID might not always be correct. Use the /REMOTE=option.

For examples of how to use this form of the %SYSLPUT statement, see “Example 1: Create a Macro Variable with %SYSLPUT (Form 1)” on page 188, “Example 2: Use the Macro Statement with %SYSLPUT (Form 1)” on page 188, and “Example 3: Mask Character Values with %BQUOTE (Form 1)” on page 189.

Copy a Group of Macro Variables (Form 2)

The %SYSLPUT statement also enables you to copy a specified group of existing macro variables from the client to the server session. The arguments used with this form enable you to define the group of macro variables to be copied based on variable type (automatic or user-defined), variable scope (global or local), and variable name. To copy all macro variables, regardless of type, scope, or name, use the _ALL_ argument in the %SYSLPUT statement.

You can also use the AUTOSIGNON system option with the %SYSLPUT statement to automatically sign on to a server session and copy specified macro variables to that server session. When the %SYSLPUT statement is specified with the AUTOSIGNON system option, the RSUBMIT command or statement automatically executes a sign-on and honors all macro variables defined in the %SYSLPUT statement for that session. For an example of using the AUTOSIGNON system option with the %SYSLPUT macro statement, see "Example 7: Use %SYSLPUT with the AUTOSIGNON Option" on page 190. For more information about the AUTOSIGNON system option, see "AUTOSIGNON" on page 99.

For examples of how to use this form of the %SYSLPUT statement to copy groups of macro variables, see "Example 4: Copy a Group of Variables to the Server Session (Form 2)" on page 189, "Example 5: Specify a Group of Variables Using LIKE= (Form 2)" on page 190, "Example 6: Overwrite Variables in the Same Server Session (Form 2)" on page 190, and "Example 7: Use %SYSLPUT with the AUTOSIGNON Option" on page 190.

Examples

Example 1: Create a Macro Variable with %SYSLPUT (Form 1)

This example creates the macro variable FLAG in the current server session and assigns a value of 1 to it.

```
%syslput flag=1;
```

Example 2: Use the Macro Statement with %SYSLPUT (Form 1)

%SYSLPUT enables you to dynamically assign values to variables that are used by macros that are executed in a server session. The macro statement %SYSLPUT is used to create the macro variable REMID in the server session and to use the value of the client macro variable RUNID. The REMID variable is used by the %DOLIB macro, which is executed in a server session, to find out which operating system-specific library assignment should be used in the server session.

Example Code 9.1 *Use %SYSLPUT to Find Out Which Libraries Can Be Used in the Server Session*

```
%macro assignlib (runid);
   signon rem&runid;
   %syslput remid=&runid;
```

```
   rsubmit rem&runid;
      %macro dolib;
         %if (&remid eq 1) %then %do;
            libname mylib 'h:';
            %end;
         %else %if (&remid eq 2) %then %do;
            libname mylib '/afs/some/unix/path';
            %end;
      %mend;
      %dolib;
   endrsubmit;
%mend;
```

Example 3: Mask Character Values with %BQUOTE (Form 1)

Because the forward slash is a macro language special character that has a special meaning to the macro processor, using it in the %SYSLPUT statement, either directly or indirectly (as a macro variable reference), will cause an error to be generated. This example uses the %BQUOTE function around the macro variable reference *&pathineed*, to mask the forward slashes in a UNIX pathname.

Example Code 9.2 *Use %BQUOTE to Mask Character Values That Are Used in a %SYSLPUT Statement*

```
  %let pathineed=/abc/xyz;
  %syslput pathineed=%bquote(&pathineed);
  rsubmit;
NOTE: Remote submit to computer commencing.
  %put &pathineed
  endrsubmit;
 %put &pathineed /abc/xyz
NOTE: Remote submit to computer complete.
```

Example 4: Copy a Group of Variables to the Server Session (Form 2)

This example uses `_ALL_` in the %SYSLPUT statement to copy two macro variables, *rc1* and *rc2*, to the server session. The %PUT statement in the RSUBMIT block uses variable references, `&rc1` and `&rc2`, to display these variables and their values in the SAS log. When the %PUT statements execute, the macro processor resolves the expressions `rc1=&rc1` and `rc2=&rc2` to `rc1=rem1` and `rc2=rem2`, respectively, and displays them in the SAS log.

```
%let rc1=rem1;
%let rc2=rem2;

%syslput _all_;
rsubmit host;
   %put rc1=&rc1
   %put rc2=&rc2
endrsubmit;
```

Example 5: Specify a Group of Variables Using LIKE= (Form 2)

By specifying _USER_ followed by LIKE=*'rc*'* in the %SYSLPUT statement below, only the user-defined macro variables whose names begin with the letters "rc" are copied to the server session. Because the macro variable *unixHost* does not meet the pattern-matching criteria, it is not recognized by the %PUT statement in the server session and a warning is displayed in the log. The %PUT statements cause the expressions rc1=&rc1 and rc2=&rc2 to be displayed as rc1=rem1 and rc2=rem2 in the SAS log.

```
signon foo sascmd="sas";
   %let rc1=rem1;
   %let rc2=rem2;
   %let unixHost=rem3;

   %syslput _user_/like='rc*' remote=host;
   rsubmit host;
      %put rc1=&rc1              /* writes rc1=rem1 to the log  */
      %put rc2=&rc2              /* writes rc2=rem2 to the log  */
      %put unixHost=&unixHost;   /* generates WARNING: Apparent
symbolic  */
                                 /*  reference UNIXHOST not
resolved.     */
   endrsubmit;
```

Example 6: Overwrite Variables in the Same Server Session (Form 2)

```
signon foo sascmd="sas";
%let rc1=rem1;
%syslput _global_/like='rc*' remote=host;
rsubmit host;
   %put rc1=&rc1
endrsubmit;

 %let rc1=changeValue;

 rsubmit host;
    %put rc1=&rc2
 endrsubmit;
```

Example 7: Use %SYSLPUT with the AUTOSIGNON Option

```
options autosignon=yes sascmd="sas";
%let rc1=rem1;
%let rc2=rem2;
%syslput _global_/like='rc*' remote=host;
```

Example 8: Use %SYSLPUT with the AUTOSIGNON Option in Multi-task Processes

```
options autosignon;
options sascmd="sas";
%let rc1=rem1;
%let rc2=rem2;
%let trc1=test1;
%let trc2=test2;
%syslput _global_/like='rc*' remote=host1;
%syslput _global_/like='trc*' remote=host2;
Rsubmit host1;
   %put rc1=&rc1;
   %put rc2=&rc2;
Endrsubmit;
Rsubmit host2;
   %put trc1=&trc1;
   %put trc2=&trc2;
Endrsubmit;
```

%SYSRPUT Statement

Assigns a value from the server session to a macro variable in the client session.

Valid in: server session

Syntax

Form 1: **%SYSRPUT** *macro-variable*=*value*;

Form 2: **%SYSRPUT** _USER_
</LIKE=‘*character-string*’>;

Syntax Description

macro-variable
specifies the name of a macro variable in the client session.

value
is a macro variable reference, a macro invocation, or a character string in the server session that is assigned to the *macro-variable* in the client session.

/LIKE=<*‘character-string’* >
specifies a subset of macro variables whose names match a user-specified character sequence, or pattern. Only this identified group of variables with names matching the pattern will be copied to the client session.

Note: The LIKE= option is not case sensitive.

'character-string'

specifies the sequence of characters, or pattern, to be used as the criteria for determining which macro variables are to be copied to the client session. Character patterns can consist of the following:

- any sequence of characters, A-Z
- any sequence of digits, 0-9
- a single wildcard character in the form of an asterisk (*)

The wildcard character (*) cannot be embedded or used more than once in the character string. The examples below illustrate how the LIKE= option works with the wildcard character. For these examples, assume that the following macro variables are defined in the client session: *rc1, rc2, linuxHOST,* and *myHOST*:

`like='rc*';`	Wildcard at the end: returns *rc1* and *rc2.*
`like='*Host';`	Wildcard at the beginning: returns *linuxHOST* and *myHOST.*
`like='*host';`	Wildcard at the beginning and lowercase "h" in name: returns *linuxHOST* and *myHOST.*
`like='r*c';`	Wildcard in the middle: is not valid and returns a syntax error.
`like='*rc*';`	More than one wildcard (at beginning and end): is not valid and returns a syntax error.
`like='rc';`	Wildcard not specified: returns nothing (no match)
`like=' ';`	Wildcard not specified and *'character-string'* is empty: returns nothing (no macro variables are copied)

Restrictions The wildcard (*) cannot be embedded in the character-string.

The wildcard (*) can be specified only once in the character-string.

Requirement The wildcard (*) must be used at either the beginning or the end of the character-string.

USER

copies all remote user-defined macro variables from the remote host to the local host at the same time.

Details

Overview

The %SYSRPUT macro statement is remotely submitted to the server session in order to assign a value that is available in the server session to a macro variable that can be accessed from the client session.

Like the %LET statement, the %SYSRPUT statement assigns a value to a macro variable. Unlike %LET, the %SYSRPUT statement assigns a value to a variable in the client session, not in the server session where the statement is executed. The %SYSRPUT statement stores the macro variable in the Global Symbol Table in the client session.

A synchronization point identifies the time (during an asynchronous RSUBMIT) at which the macro variable that is specified in the %SYSRPUT statement is defined to the client session and is available for execution in the client session.

Synchronization Points

Here are the three possible synchronization points:

1. when the RGET statement is executed.

   At this time, all macro variables that were specified by using %SYSRPUT are defined in the client session and are available for execution.

2. when a synchronous RSUBMIT is started in the same server session that an asynchronous RSUBMIT is already running in.

3. when the SIGNOFF statement is executed.

   All macro variables that were specified using %SYSRPUT are defined in the client session and are available for execution.

All currently spooled log and output statements are retrieved and merged into the client Log and Output windows. RSUBMIT continues from then on as if it were synchronous. Control is returned to the client session after the RSUBMIT has completed. In addition, %SYSRPUT macro variables that have been generated during the asynchronous RSUBMIT up to that point are defined in the client session. Thereafter, RSUBMIT becomes synchronous, and macro variables are synchronized immediately when they are executed.

To override the default for an asynchronous RSUBMIT, you can specify the SYSRPUTSYNC= option in the RSUBMIT statement. Macro variables are set at the time of execution rather than at a synchronization point in the client session.

Examples

Example 1: %SYSRPUT

The %SYSRPUT statement is useful for capturing the value that is returned in the SYSINFO macro variable and for passing that value to the client session. The

SYSINFO macro variable contains return-code information that is provided by SAS procedures.

This example shows how to download a file and to return information about the success of the step from a batch job.

Example Code 9.3 *Using %SYSRPUT to Find Out Whether a Download Is Successful*

```
signon rhost;
rsubmit;
    proc download data=remote.mydata
         out=local.mydata;
   run;
   %sysrput retcode=&sysinfo;
endrsubmit;
%macro checkit;
   %if &retcode=0 %then %do;
      code-to-be-executed-in-client-session
   %end;
%mend checkit;
%checkit;
```

The %SYSRPUT statement occurs after a PROC DOWNLOAD statement. The value that is returned by &SYSINFO indicates the success of the PROC DOWNLOAD statement. After execution in the server session has completed, the value of the return code that is stored in RETCODE is checked. If server execution is successful, execution continues in the client session.

If SIGNON, RSUBMIT, or SIGNOFF fails, a SAS/CONNECT batch job returns a nonzero system condition code. To find out the status of an RSUBMIT execution, use the %SYSRPUT statement. This macro checks the value of the automatic macro variable SYSERR. For details, see *SAS Macro Language: Reference*.

Example 2: %SYSRPUT

This example shows the execution of an asynchronous RSUBMIT. The SYSRPUTSYNC= option is specified in order to set the client session's macro variable when %SYSRPUT executes rather than when a synchronization point is encountered. The value of the macro variable STATUS can be checked to monitor the progress of the asynchronous RSUBMIT.

Example Code 9.4 *Using %SYSRPUT to Monitor the Progress of an Asynchronous RSUBMIT*

```
rsubmit wait=no csysrputsync=yes;
   %sysrput status=start;
   proc download inlib=sales outlib=tmp;
   run;
   %sysrput status=salescomplete;
   proc download inlib=inventory outlib=tmp;
   run;
   %sysrput status=inventorycomplete;
   proc upload data=sales.store10;
   run;
   %sysrput status=storecomplete;
endrsubmit;
```

Example 3: %SYSRPUT

This example shows how to identify the server session that the client session is signed on to:

```
rsubmit;
%sysrput rhost=&sysscp;
endrsubmit;
```

WAITFOR Statement

Causes the client session to wait for the completion of one or more tasks (asynchronous RSUBMIT statements) that are in progress.

Valid in: client session

Syntax

WAITFOR <_ANY_|_ALL_> *task task2*… <TIMEOUT=*seconds*>;

Syntax Description

ANY
causes the client session to wait for the completion of any of the specified tasks (a logical OR of the completion task states).

ALL
causes the client session to wait for the completion of all of the specified tasks (a logical AND of the completion task states).

task...taskn
identifies one or more asynchronous tasks to be completed. The task corresponds with the *server–ID* that is associated with the CONNECTREMOTE= option when the RSUBMIT is submitted.

TIMEOUT=*seconds*
allots the interval, in seconds, to wait for one or more asynchronous tasks to complete. If the specified tasks have not completed by time-out, then the WAITFOR statement is terminated, control is returned to the client session, and the asynchronous tasks continue to execute until they are completed. The SYSRC system macro variable will have a nonzero status.

If the specified tasks are completed before time-out, the WAITFOR statement returns control to the client session as soon as the specified tasks are completed.

Note: Specifying TIMEOUT=0 is equivalent to providing no TIMEOUT value. Specifying a value of 0 causes the client session to wait indefinitely for the asynchronous tasks to complete before control is returned to the client session.

Default 0

See “CONNECTREMOTE=” on page 108

Details

The WAITFOR statement causes the client session to wait for the completion of one or more tasks that are in progress in the server session as specified by the options _ANY_ or _ALL_. WAITFOR synchronizes dependent tasks. You can use WAITFOR only for asynchronously executing tasks. If you use WAITFOR and there are no asynchronous tasks executing, the WAITFOR statement does not enforce a wait condition. Instead, execution continues in the client session.

The name of the task corresponds with the *server-ID*.

The WAITFOR statement can wait for the completion of one or more tasks. If more than one task is specified and neither _ANY_ nor _ALL_ is specified, _ANY_ is implied. The client session will wait for any of the listed tasks to complete before resuming control. This is not an error condition.

If more than one task is specified, and the _ANY_ option is specified, then the client session waits for the completion of any of the specified tasks (a logical OR of the completion task states). If the _ALL_ option is specified, the client session waits for the completion of all the specified tasks (a logical AND of the completion task states). The WAITFOR statement does not support complex logical statements, such as A OR (B AND C).

Invalid tasks that are specified in the WAITFOR statement are ignored but are identified in notes in the SAS log.

Examples

Example 1: Example 1: WAITFOR

The following example shows the suspension of the client session until both tasks have completed or 300 seconds (5 minutes) pass, whichever occurs first.

```
waitfor _all_  remhost printjb  timeout=300;
```

Example 2: Example 2: WAITFOR

The following WAITFOR statement causes the client session to wait for either the REMHOST or FORMATJB task to complete.

```
waitfor _any_ remhost formatjb;
```

LISTTASK Statement

Lists all active connections or tasks and identifies the execution status of each connection or task.

Valid in: client session

Syntax

LISTTASK <<_ALL_> | <*task*>|>;

Syntax Description

ALL
provides status information about all current tasks.

task
provides status information for the specified task. Identifies the specific task by a name that corresponds to the *server-ID* that is associated with the CONNECTREMOTE= option in the RSUBMIT or SIGNON statement or command.

See

Details

The LISTTASK statement lists information about all tasks in the current server session or about a single active task by name. If neither _ALL_ nor *task* is specified, information about all current tasks is listed.

Examples

Example 1: Example 1: LISTTASK

The following LISTTASK statement lists information for all tasks. The appearance of the output varies by operating environment.

```
listtask _all_;
"REMHOST" - - - - - - - - -
          Type: SAS/CONNECT Process
          State: RUNNING ASYNCHRONOUSLY
"TASK1" - - - - - - - - - -
          Type: SAS/CONNECT Process
          State: COMPLETE
```

Example 2: Example 2: LISTTASK

The following LISTTASK statement lists information for the REMHOST task only. The appearance of the output varies by operating environment.

```
listtask remhost;
"REMHOST" - - - - - - - - - -
          Type: SAS/CONNECT Process
          State: COMPLETE
```

KILLTASK Statement

For asynchronous tasks, forces one or more active tasks or server sessions to terminate immediately.

Valid in: client session

Syntax

KILLTASK _ALL_ | *task1...taskn*;

Syntax Description

ALL
terminates all active asynchronous tasks.

task
terminates a specific task by a name that corresponds to the *server-ID* that is associated with the CONNECTREMOTE= option in the RSUBMIT statement.

Restriction Use the KILLTASK statement only when executing an asynchronous RSUBMIT.

See “CONNECTREMOTE=” on page 108

Details

The KILLTASK statement enables users to terminate one or more tasks or server sessions that are executing asynchronously. The KILLTASK statement is useful only for an asynchronous RSUBMIT.

Note: KILLTASK should be used for asynchronous tasks that seem to be hung or to be having a problem. KILLTASK ends the server session. However, do not substitute KILLTASK for SIGNOFF. Use SIGNOFF to terminate server sessions that are functioning normally.

KILLTASK causes any log or output lines, as applicable, that have accumulated in the backing store to be sent to the parent Log and Output windows. Before the data is sent to the parent Log and Output windows, this message is displayed:

```
NOTE: Process TASK1 was terminated by KILLTASK statement.
```

KILLTASK removes the specified task from the list of active tasks and from the LISTTASK output. If KILLTASK is executed for a completed task, this message is displayed and the task will not be terminated:

```
NOTE: Transaction TASK2 was not killed because it is not running asynchronously.
```

Task termination also deletes the content of the Work library of the server session.

Comparisons

After you use the KILLTASK statement to kill a server session that runs under z/OS, you must also sign off from the server session. If you do not also sign off from the server session, your user ID will still be connected to the server session. Here are the methods for signing off a server session:

- From the same SAS session from which you issued the KILLTASK statement, sign on to the server session, using your user ID. Then, sign off. The most recently accessed server session is assumed, by default.

  ```
  signon user-ID;
  signoff user-ID;
  ```

- Log on to your user ID, and then cancel the user ID using the CANCEL option.
- Request that an operator cancel your TSO session.

Consult your z/OS documentation for details about logging on and logging off the z/OS operating environment.

10

FILENAME Statement

Dictionary

FILENAME Statement

Associates a SAS fileref with an external file.

Valid in: client and server session

See: "FILENAME: Windows" in *SAS Companion for Windows*, "FILE Statement: UNIX" in *SAS Companion for UNIX Environments*, and "FILENAME Statement: z/OS" in *SAS Companion for z/OS*.

Syntax

FILENAME *fileref* '*filespec*' <*access-method*> <*operating-environment-options*>;

Optional Arguments

fileref
specifies the name of a file reference to an external file.

'*filespec*'
specifies the physical name of an external file so that the external file is recognized by the operating environment.

access-method
specifies a remote file access via a specific access method. For details, see the access methods that are supported in the FILENAME statement in *SAS DATA Step Statements: Reference*.

operating-environment-options
specifies details, such as file attributes and processing attributes, that are specific to the operating environment.

Details

Overview of the FILENAME Statement

The FILENAME statement associates a SAS *fileref* (a file reference name) with a *filespec*. The fileref must conform to SAS naming rules. The form of the filespec varies according to operating environment. Some environments require a fully qualified filename; other environments might permit partial pathnames.

Filerefs are a shorthand method for specifying a file in SAS statements and commands. After you define a fileref, you can use the fileref in place of the longer file specification to reference the file throughout a SAS session or program.

A fileref remains associated with an external file only for the duration of the SAS session. The association is not permanent. Also, a fileref must be defined and the FILENAME statement must be executed before a SAS statement or command that uses the fileref can execute.

Use the FILENAME RLINK Statement for Script Files

A common use of the FILENAME statement is to define filerefs for SAS/CONNECT script files. A script's fileref can then be specified in SIGNON and SIGNOFF commands to identify the SAS/CONNECT script that starts or ends the connection.

You can define a default fileref for a script file in a FILENAME statement. The default script fileref is RLINK. If you specify RLINK as the fileref for your script, you do not need to specify a fileref or a filespec in SIGNON and SIGNOFF commands or statements. When SAS executes a SIGNON or a SIGNOFF command without a specified fileref or a filespec, SAS automatically searches for a file that is defined with RLINK as the fileref. If RLINK has been defined, SAS executes the corresponding script.

Use a FILENAME Statement in the SAS Autoexec File

You can insert a FILENAME statement in the SAS autoexec file to automatically start and end a SAS/CONNECT server session. An *autoexec file* contains SAS statements and commands that you set up to execute automatically each time you invoke SAS. Its purpose is to automate the execution of statements, commands, and entire programs that you use routinely in SAS processing. If you use an autoexec file that contains a FILENAME statement that defines your script's fileref,

you do not have to enter and execute the FILENAME statement each time you want to establish a connection.

For details about setting up an autoexec file, see the appropriate SAS Companion documentation for your environment and *SAS Language Reference: Concepts*.

Use a FILENAME Statement with the UPLOAD and DOWNLOAD Procedures

You can combine the FILENAME statement with the UPLOAD and DOWNLOAD procedures to copy external files between SAS sessions. For example, in the client session, use the FILENAME statement to assign a fileref. The fileref defines the target location for the external file copy. In the server session, use the FILENAME statement to assign a fileref to the file to be downloaded to the client session.

Examples

Example 1: Use a FILENAME Statement for a Script File

If a SAS/CONNECT script is written and copied to a directory in your client environment, you could use the FILENAME statement to define the default fileref RLINK for the script, as follows:

```
filename rlink 'external-file-name';
```

Because you defined RLINK as the script's fileref, you can use the shortest form of the SIGNON and SIGNOFF commands or statements. For example, to start the connection, enter the following:

```
signon;
```

If you use one script to start the connection and another script to end the connection, you must define a unique fileref for each script. For example:

```
filename rlink 'start-link-script-file';
filename endit 'end-link-script-file';
```

Subsequently, to start the connection, enter the following command or statement, which uses the default fileref RLINK for the sign-on script:

```
signon;
```

To end the connection, enter the following:

```
signoff endit;
```

Example 2: Use a FILENAME Statement with the UPLOAD and DOWNLOAD Procedures

Suppose you want to download an external file from a server session to a client session that runs in a directory-based operating environment. Submit the following FILENAME statement to assign the fileref in the client session:

```
filename lhost 'client-file-name';
```

Then remotely submit the following statements to assign the fileref in the server session and to perform the download:

```
rsubmit;
filename rhost 'server-file-name';
   proc download infile=rhost outfile=lhost;
   run;
endrsubmit;
```

11

LIBNAME Statement

Dictionary

LIBNAME Statement

Associates a libref (a shortcut name) with a SAS library that is located on the server for client access.

Valid in:	client session
Category:	Data Access
Operating environment:	"LIBNAME Statement: UNIX" in *SAS Companion for UNIX Environments*, "LIBNAME: Windows" in *SAS Companion for Windows*, and "LIBNAME Statement: z/OS" in *SAS Companion for z/OS*.
See:	Base SAS "LIBNAME" in *SAS Global Statements: Reference*.

Syntax

LIBNAME *libref* <*engine*> <'*SAS-library*'> SERVER=*server-ID* <*options*> <*engine/operating environment-options*>;

Required Arguments

libref
specifies the name of a library reference to a SAS library that is located on the server. The libref that you specify is presumed to be the server libref for an existing server library. As alternatives, you could use the SLIBREF= option or the physical name of the data library.

The *libref* that you specify must be a valid SAS name, and it must be the first argument in the LIBNAME statement.

'SAS-library'
specifies the physical name for the SAS library on the server to access. If you specify a server library either as the libref or as the value for the SLIBREF= option, you must omit the physical name.

If you specify *'SAS-library'*, the name must be a valid physical name, and it must be enclosed in single or double quotation marks. For details about specifying a SAS library, see the documentation that is appropriate to your operating environment.

SERVER=*server-ID*
specifies the ID of the server (where the SAS library is located) that you previously signed on to. The *server-ID* is the value of the *remote-session-ID* that is specified in the SIGNON statement on page 127. A server name must be 8 characters or less and start with an alphabetic character. To specify a server name that contains more than eight characters, you must store the name in a macro variable.

Do not use the <*computer-name.port-number*> format to specify the <*server-ID*> value in the SIGNON statement if you are going to specify a LIBNAME statement on the server. Instead, use the <*computer-name._ _port-number*> format for the *server-ID* value in both the LIBNAME statement and the SIGNON statement.

```
signon hrcomp1._ _2267;
libname myLib server=hrcomp1._ _2267;
```

Optional Arguments

ACCESS=READONLY
controls a client's Read access to a SAS library on the server. If you specify this option, you can read but not update data in the library.

engine
specifies the name of a valid SAS engine for a client to access the server library. You should not use this option because the client automatically determines which engine to use for accessing a server. Specify this option only to override the SAS default for a specific server, or to reduce the time that is needed to determine which engine to use to access a specific server.

For example, if the server library is located on a server that is running SAS 9 or later, you could specify the REMOTE engine. Specifying an explicit engine might improve performance slightly.

For a list of valid engines, see the SAS documentation for your operating environment. For background information about engines, see *SAS Language Reference: Concepts*.

The *engine* argument is positional. If you use it, it must follow the libref.

CAUTION
Do not confuse the ENGINE argument with the RENGINE= option. An engine is used by a client to access a server. The RENGINE= option is used by the server to access its SAS library.

SLIBREF=*server-libref*
specifies an existing server libref that you want to reference from the client. Use this option when you want to reference an existing server libref, but you want to use a different name for that libref on the client. If you specify the SLIBREF= option, you do not need to specify the physical name for the SAS library on the server. SLIBREF= *server-libref* and *'SAS-library'* are mutually exclusive.

Engine and Operating Environment Options

RENGINE=*engine-name*
specifies the engine for the server session to use to access the SAS library on the server. Using this option is usually unnecessary because the server automatically determines the engine to use for processing the data library. Specify this option only to override the SAS default for a specific library, or to reduce the time that is used by the server to determine the engine to use.

CAUTION
Do not confuse the RENGINE= option with the ENGINE argument. The RENGINE= option is used by the server to access its SAS library. The ENGINE argument is used by a client to access a server.

ROPTIONS="*option=value<option=value>* ..."
specifies remote options and options that are specific to an operating environment, which the client passes to the engine on the server that processes the SAS library. ROPTIONS can be specified for either the default engine or an alternative engine that is specified by using the RENGINE= option. You can specify one or more options in the form *option=value*. Use a blank to separate the options. You can use the ROPTIONS= option to pass any valid option for the targeted engine. For information about the options that are supported by a specific engine, see the documentation for the engine that you use. For details about options that are specific to an operating environment, see the documentation that is appropriate for your operating environment.

RMTVIEW=YES | NO
determines whether SAS views are interpreted in the server session or the client session. SAS views include DATA step views, in addition to views that are created by using the SQL procedure and the ACCESS procedure (in SAS/ACCESS software).

SAS views, like SAS data sets, are accessed through an engine. Where a SAS view is interpreted determines where the view engine is loaded and used. DATA step views use the SASDSV engine, and PROC SQL views use the SQLVIEW engine. SAS creates a product-specific engine for each SAS/ACCESS interface product that the SAS/ACCESS views use for that interface.

When SAS views are interpreted in the server session, the server session might require large amounts of processor time and storage. However, the amount of data that is transferred to the client session might be reduced. Conversely, preventing view processing in the server session might increase the amount of data that is transferred between the server and the client, but minimizes server processing time.

Setting the RMTVIEW= option to NO causes SAS views to be interpreted at the client.

Default YES, which causes views to be interpreted in the server session.

Examples

Example 1: Assign and Define a Libref to Access a Library on a Server

The following statement associates the libref **Sqldslib** with the SAS library **Sasxyz.Viewlib.Sasdata**. This library is accessed through the server MVSHOST, which is running in a server session.

```
libname sqldslib 'sasxyz.viewlib.sasdata' server=mvshost;
```

Example 2: Associate a Client Libref with a Server Libref

The following statement associates the client libref Applib with the server libref Servlib. This library is accessed through the server MYHOST.

```
libname applib slibref=servlib server=myhost;
```

Example 3: Specify a Server in the LIBNAME Statement

The following example shows a spawner invocation on a computer named MYHOST.MY.NET.WORK. The -SERVICE option specifies that the spawner listens for client connections on port 2323.

```
cntspawn -c tcp -service 2323
```

In the following example, a client uses the TCP/IP access method to connect to a server session by using a spawner. The name of the computer that the spawner runs on and the number of the port that the spawner listens on are assigned to the macro variable REMNAME.

Note: Use a space to separate the computer name from the port number.

A client signs on to the server at the specified port that is defined by REMNAME. The LIBNAME statement establishes the libref ScorCard to point to a library via the server and port that are defined by REMNAME.

```
options comamid=tcp;
%let remname=myhost.my.net.work 2323; /* space between computer */
signon remname;                       /* name and port number
libname scorcard '.' server=remname;
```

12

LIBNAME Statement, SASESOCK Engine

Dictionary

LIBNAME Statement: SASESOCK Engine

Associates a libref with a TCP/IP pipe (instead of a physical disk device) for processing input and output. The SASESOCK engine is required for SAS/CONNECT applications that implement MP CONNECT with piping.

Valid in:	client session and server session
Category:	Data Access
Operating environment:	"LIBNAME Statement: UNIX" in *SAS Companion for UNIX Environments*, "LIBNAME: Windows" in *SAS Companion for Windows*, and "LIBNAME Statement: z/OS" in *SAS Companion for z/OS*.
See:	Base SAS "LIBNAME" in *SAS Global Statements: Reference*

Syntax

LIBNAME *libref* SASESOCK "*port-specifier*" <TIMEOUT=*time-in-seconds*>;

Required Arguments

libref
specifies a reference to a TCP/IP pipe instead of to a physical disk device.

The *libref* that you specify must be a valid SAS name, and it must be the first argument in the LIBNAME statement.

SASESOCK "*port-specifier*"
identifies the SASESOCK engine to process input to and output from a TCP/IP port instead of a physical disk device.

"*port-specifier*" can be represented in these ways:

":*explicit-port*"
is a hardcoded port number that specifies an explicit port on the computer where the asynchronous RSUBMIT is executing. The port number specified must be between 1 and 65,535.

Example:

```
LIBNAME payroll SASESOCK ":256";
```

Range 1–65,535

Requirement If the port number that you specify is in use, access will be denied until it is available again.

":*port service*"
specifies the name of the port service on the computer where the asynchronous RSUBMIT is executing.

Example:

```
LIBNAME payroll SASESOCK ":pipe1";
```

Requirements If you specify a port service, it must be configured in the SERVICES file of the computers at which the client and server sessions are running.

If the port service that you specify is in use, access will be denied until it is available again.

See For details about configuring port services in the SERVICES file, see “Configure the TCP/IP Services File” on page 309.

"*computer-name*:*port-number*"
specifies an explicit port number on the computer that is specified by *computer-name*.

Example:

```
LIBNAME payroll SASESOCK "apex.finance.com:256";
```

Requirement If the port number that you specify is in use, access will be denied until it is available again.

"*computer-name*:*port service*"
specifies the name of the port service on the computer that is specified by *computer-name*.

Example:

```
LIBNAME payroll SASESOCK "apex.finance.com:pipe1";
```

Requirements If you specify a port service, it must be configured in the SERVICES file of the computers at which the client and server sessions are running.

If the port service that you specify is in use, access will be denied until it is available again.

See For details about configuring port services in the SERVICES file, see "Configure the TCP/IP Services File" on page 309..

"*implicit-port*"
is an alias that refers to an implicit port number that SAS dynamically selects from a pool of available ports when the asynchronous RSUBMIT begins execution. The actual port that SAS selects is stored automatically in the SAS Metadata Server without your knowledge of the port's identity. Because the alias is mapped to the port and is stored in the metadata server, you can always use the alias without concern about the actual port number.

Example:

```
LIBNAME payroll SASESOCK "mypipe";
```

If you use an alias that specifies an implicit port, the client and server sessions must have access to the SAS Metadata Server. The port number that is assigned to the alias that you specify is stored in the SAS Metadata Server. To have access to a SAS Metadata Server, several metadata properties must be configured via selected SAS options in the SAS session. Here is an example:

```
options metaserver="a123.us.company.com"
        metaport=9999
        metauser="metaid"
        metapass="metapwd"
        metaprotocol=bridge
        metarepository="myrepos";
```

Requirements

If you use an implicit port, do not configure the alias in the SERVICES file.

See If you specify an implicit port, see SAS system options METASERVER, METAPORT, METAUSER, METAPASS, METAPROTOCOL, and METAREPOSITORY in *SAS Language Interfaces to Metadata*.

Optional Argument

TIMEOUT=*time-in-seconds*
specifies the amount of time, in seconds, that a SAS process will wait to successfully connect to another process. The value for *time-in-seconds* should be a positive integer that does not contain symbols, such as +, commas, or decimal points. Valid *time-in-seconds* values are 1 to 86,400, inclusively. Negative values, zero, and non-numeric values will generate a warning and set the time-out to 10 seconds.

Default 10

Range 1–86400, inclusive

See For an explanation of MP CONNECT using piping, see “Pipeline Parallelism” on page 33.

For an example of a SAS/CONNECT application that implements MP CONNECT using piping, see “Example 6: MP CONNECT with Piping” on page 64.

Example

```
libname in1 sasesock ":pipe1" timeout=50;
```

13

Commands

Dictionary

SIGNON Command

Initiates a connection between a client session and a server session.

Valid in: client

Syntax

SIGNON <*options*>

Details

For more details, see “SIGNON” on page 127

SIGNOFF Command

Ends the connection between a client session and a server session.

Valid in: Client session

Syntax

SIGNOFF <*options*>

Details

For more details, see “SIGNOFF” on page 147

RDISPLAY Command

Creates a Log window to display the lines from the log and an Output window to list the output generated from the execution of the statements within an asynchronous RSUBMIT block.

Valid in: client session

Syntax

RDISPLAY <CONNECTREMOTE=*server-ID*>

Details

For more details, see “RDISPLAY” on page 182

RSUBMIT Command

Marks the beginning of a block of statements that a client session submits to a server session for execution.

Valid in: client session

Syntax

RSUBMIT <*options*>

Details

For more details, see “RSUBMIT” on page 161

Chapter 14

UPLOAD Procedure

Overview: UPLOAD Procedure

Introduction

After a SAS/CONNECT client connects to a SAS/CONNECT server, you can transfer files between a client session and a server session by using the UPLOAD procedure.

Using PROC UPLOAD in SAS/CONNECT, you can do the following:

- transfer multiple SAS files in a single step by using the INLIB= and OUTLIB= options. This capability enables you to transfer an entire library or selected members of a library in a single PROC UPLOAD step.
- upload specific entries in a catalog or specific members in a library by using the SELECT and EXCLUDE statements.
- use WHERE processing and SAS data set options when uploading individual SAS data sets.
- replicate selected data set attributes when uploading a data set.
- transfer data sets and catalog entries that have been modified on or after the specified date.
- specify which translation table should be used when uploading a SAS catalog.

See Chapter 5, “Using Data Transfer Services,” on page 87 for more information about using data transfer services with SAS/CONNECT.

Syntax: UPLOAD Procedure

PROC UPLOAD
<*data-set-options*>
<*catalog-options*>
<*library-options*>
<*external-file-options*>
<AFTER=*date*>
<CONNECTSTATUS=YES | NO>;
WHERE *where-expression-1* <*logical-operator where-expression-n*>;
EXCLUDE *list* </MEMTYPE=*mtype* | ENTRYTYPE=*etype*>;

SELECT </MEMTYPE=*mtype* | ENTRYTYPE=*etype*>;

TRANTAB NAME=*translation-table-name* <TYPE=(*etype-list*)> <OPT=DISP | SRC>;

PROC UPLOAD Statement

Transfers files from the client to the server.

Alias: none

Syntax

PROC UPLOAD

<*data-set-options*>
<*catalog-options*>
<*library-options*>
<*external-file-options*>
<AFTER=*date*>
<CONNECTSTATUS=YES | NO>;

Data Set Options

CAUTION

Do not confuse the PROC UPLOAD data set options with the SAS data set options. The PROC UPLOAD data set options are valid only in the context of PROC UPLOAD. However, two of the PROC UPLOAD data set options (DATA= and OUT=) can be further characterized by SAS data set options. For details, see the descriptions for the DATA= on page 222 option and the OUT= on page 227 option.

data-set-options can be one or more of the following:

- "CONSTRAINT=YES | NO" on page 221
- "DATA=*client-SAS-data-set* <(*SAS-data-set-options*)>" on page 222
- "DATECOPY" on page 222
- "EXTENDSN=YES | NO" on page 222
- "INDEX=YES | NO" on page 223
- "OUTLIB=*server-SAS-data-set* <(*SAS-data-set-options*)>|OUT=" on page 227
- "V6TRANSPORT" on page 228
- "XATTR=YES | NO" on page 229

Catalog Options

catalog-options can be one or more of the following:

- "ENTRYTYPE=*etype* " on page 222

- "EXTENDSN=YES | NO" on page 222
- "INCAT=*client-SAS-catalog*" on page 223
- "OUTCAT=*server-SAS-catalog*" on page 226

Library Options

library-options can be one or more of the following:

- "CONSTRAINT=YES | NO" on page 221
- "EXTENDSN=YES | NO" on page 222
- "GEN=YES | NO" on page 223
- "INDEX=YES | NO" on page 223
- "INLIB=*client-SAS-library* " on page 226
- "MEMTYPE=(*mtype-list*) " on page 226
- "OUTLIB=*server-SAS-library* " on page 228
- "VIEWTODATA" on page 228
- "V6TRANSPORT" on page 228

External File Options

external-file-options are the following:

- "BINARY" on page 221
- "INFILE=*client-file-identifier*" on page 224
- "OUTFILE=*server-file-identifier*" on page 226

Optional Arguments

AFTER=*date*
specifies a modification date in the form of a numeric date value or a SAS date constant.

This option is valid for transferring data sets, catalogs, and libraries. Its use results in data sets or catalog entries being transferred only if they have been modified on or after the specified date.

The AFTER= option is also valid for external file transfers between most computers. If a computer is unable to perform the transfer, this message is displayed:

```
ERROR: AFTER= not supported on this platform.
NOTE: The SAS System stopped processing this step
      because of errors.
```

Note: The AFTER= option is available in SAS 6.09E, SAS 6.11 TS040, and later.

For example, the following statement causes the transfer of any data sets or catalog entries in the library Accts only if they have been modified on or after December 30, 2001.

```
proc upload inlib=accts outlib=accts
    after='30dec01'd status=no;
```

If your client session is using an earlier release of SAS that does not support this option, PROC UPLOAD produces the following message:

```
Warning: AFTER= option not supported by earlier
         release; option will be ignored.
```

Note: If the client is running SAS 6.11 TS020 or SAS 6.08 TS415 through SAS 6.08 TS430, the option is ignored, but no warning is displayed.

BINARY

specifies an upload of a binary image (an exact copy) of an external client file. Use this option only for uploading external files.

Note: External files are files that are not SAS files.

By default, if the client and server run in different operating environments (for example, UNIX and Windows), PROC UPLOAD transfers a file from the client to the server, translating the file from UNIX representation to Windows representation. Furthermore, PROC UPLOAD inserts record delimiters that are appropriate for the target environment.

You do not always want to translate a file. For example, you might need to upload executable files from the client to the server and later download them to the same or a different client. Binary file format also conserves resources for users who store their own files and for system backups. The BINARY option prevents delimiters from being inserted for each file record that is created at the server. In addition, if the client and server use a different method of data representation, the BINARY option prevents any data translation between ASCII and EBCDIC.

Example "Example 16: Distribute an .EXE File from the Server to Multiple Clients: UPLOAD" on page 247.

CONSTRAINT=YES | NO

specifies if integrity constraints should be re-created on the server when a SAS data set that has integrity constraints defined is uploaded. You can specify this option with the DATA= option (if you omit the OUT= option) or with the INLIB= and OUTLIB= options.

By default, integrity constraints are re-created only when you upload a SAS library or when you upload a single SAS data set and omit the OUT= option. If you specify the OUT= option with the DATA= option, the integrity constraints are not re-created.

CONNECTSTATUS=YES | NO

specifies whether the Transfer Status window should be displayed during a transfer. By default, the UPLOAD procedure displays the "Transfer Status Window" on page 92 (CONNECTSTATUS=YES)

Alias CSTATUS=, STATUS=

Default YES

DATA=*client-SAS-data-set* <(*SAS-data-set-options*)>
specifies a SAS data set to upload from the client to the server. If the data set is a permanent SAS data set, you must define a libref before the PROC UPLOAD statement and specify the two-level name of the data set.

If you specify the name of a data view in the DATA= option, the materialized data is uploaded to the server, not to the view definition.

If you do not specify the DATA=, INCAT=, INLIB=, or INFILE= option, the last SAS data set that was created on the client during your SAS session is uploaded.

Requirement When you specify the DATA= option, you must either specify the OUT= option or omit all other output file options.

Interaction The data set is characterized by SAS data set options that were specified when the data set was created. For example, specifying the COMPRESS=YES data set option would cause all observations in the data set to be compressed. You use SAS data set options to change the data set's characteristics or to apply new characteristics.

See "OUTLIB=*server-SAS-data-set* <(*SAS-data-set-options*)>|OUT=" on page 227

SAS Data Set Options: Reference

Example "Specify Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 231

DATECOPY
retains the date on which a SAS data set was created and the date on which a SAS data set was last modified for each data set that is transferred.

ENTRYTYPE=*etype*
specifies a catalog entry type to be uploaded. Examples of catalog entry types include DATA and FORMAT.

Alias ETYPE=, ET=

Requirement To use this option, you must also specify the INCAT= and OUTCAT= options.

EXTENDSN=YES | NO
specifies whether to promote the length of short numerics (length less than 8 bytes) when transferring.

NO
indicates that the length of numeric variables is not promoted.

YES
indicates that 1 will be added to the length of any numeric variable that has a length of less than 8 bytes before it is transferred to the server.

The behavior of the EXTENDSN= option varies according to the SAS release that is used.

- If both the client and the server run SAS 8 or a later release, and the V6TRANSPORT option is specified, then the default is to promote the length

of a numeric variable whose length is less than 8 bytes. This is consistent with SAS 6 behavior. To override this behavior, specify EXTENDSN=NO along with the V6TRANSPORT option in the UPLOAD statement.

- If either the client or the server runs SAS 6, neither the V6TRANSPORT nor the EXTENDSN= option is supported or recognized.
- If the client runs SAS 6 and the server runs SAS 8 or a later release, a numeric variable whose length is less than 8 bytes is promoted , by default. In this case, specify EXTENDSN=NO in order to override the SAS 6 default and to prevent the promotion.

See "File Format Translation Algorithms" on page 462 for information about translating file formats between a client and server that run on computers whose internal representations are incompatible.

Default NO

GEN=YES | NO
specifies that data set generations are to be sent during library transfers.

YES
specifies that data set generations are sent during library transfers.

NO
specifies that data set generations are not sent during library transfers.

Default YES

INCAT=*client-SAS-catalog*
names a SAS catalog that you want to upload from the client to the server. If the catalog is stored in a permanent SAS library, you must define a libref before specifying the PROC UPLOAD statement, and you must specify the catalog's two-level name.

To upload all of the catalogs in a SAS library, specify INCAT=*libref*._ALL_.

If you specify this form for the INCAT= option, you must specify the same form for the OUTCAT= option.

You can transfer catalogs with entries that contain graphics output as well as other catalog entries.

CAUTION
Some catalog entry types are not compatible between SAS releases. If you attempt to upload a catalog entry from a client to a server and they run different SAS releases, then the client catalog entry that is being uploaded might not be supported at the server. In this case, the catalog entry will not be transferred and the following error message is displayed:

```
WARNING: FILEFMT entries
```

INDEX=YES | NO
specifies whether to allow for the upload or download of indexes that are defined on a SAS data set. This option is turned on by default (set to YES) in PROC UPLOAD and PROC DOWNLOAD. The INDEX=YES option is invalid when the OUT= option is specified. If INDEX=YES is specified with the OUT= option, then INDEX=YES is ignored and a WARNING is sent to the SAS log.

To re-create an index on the server, you can specify INDEX=YES when using the DATA= option (if you omit the OUT= option) or when using the INLIB= and

OUTLIB= options. Indexes are re-created with the INDEX= procedure option only when you upload a SAS data set and omit the OUT= option.

An index will be re-created in the server session by default under these conditions:

- if you do not specify the INDEX= option, you upload a single data set, and you omit the OUT= option in PROC UPLOAD
- if you do not specify the INDEX= option, and you upload an entire SAS library

For information about PROC UPLOAD options and the default behavior of data set options on data sets being transferred, see Table 14.9 on page 232.

Do not confuse the PROC UPLOAD data set option, INDEX=, with the SAS data set option, INDEX=. Both options can be used in the PROC UPLOAD statement, but they have different roles. The INDEX=<*data-set-name*> option is used in the OUT= statement of PROC UPLOAD to create an index on the server data set during the upload.

The INDEX=`YES`|`NO` data set option is a PROC UPLOAD procedure data set option that is used to allow or deny the upload of an existing index.

Default YES

Restriction If the INDEX=YES and the OUT= option are used together in a PROC UPLOAD statement, indexes defined on the DATA= data set will not be re-created on the server.

Requirement If you choose to re-create an index for the data set being uploaded (using the INDEX= data set option), you must specify one or more variables to be indexed.

See For syntax information about the SAS data set option INDEX=, see "INDEX= Data Set Option" in *SAS Data Set Options: Reference*.

For conceptual information about SAS data set indexing, see "Understanding SAS Indexes" in *SAS Language Reference: Concepts*.

Example "Example 18: Create an Index with OUT= Using PROC UPLOAD" on page 248.

INFILE=*client-file-identifier*
specifies the external file that you want to upload to the server from the client.

If you use the INFILE= option, you must also use the OUTFILE= option.

client-file-identifier can be one of the following:

fileref
is used if you have defined a fileref on the client that is associated with a single file. You must define the fileref before specifying the PROC UPLOAD statement.

fileref(member)
is used if you have defined a fileref on the client that is associated with an aggregate storage location, such as a directory.

member
specifies one or more files in that aggregate storage location. You can use the asterisk character (*) as a wildcard in the *member* specification to

upload multiple files via a single PROC UPLOAD statement. The * matches zero or more characters.

You must define the fileref before specifying the PROC UPLOAD statement.

Note: The transfer of hidden files is not supported when using the (*) wildcard

The following examples demonstrate the use of the wildcard character. The fileref in the examples is `loc`.

Table 14.1 *Examples: The Wildcard Character in PROC UPLOAD*

`infile=loc('*')`	A single asterisk specifies all of the files in the aggregate location.	`all files`
`infile=loc('*dat')`	A leading asterisk specifies all files that end with the same characters. The example selects all files that end with `dat`.	`testfile.dat` `report.old.dat`

`infile=loc('test*')`	A trailing asterisk specifies all files that begin with the same characters.
	The example selects all files that begin with `test`.
		`test.dat`
		`testfile.history`
		`test.tar.gz`

`infile=loc('t*file')`	An embedded asterisk specifies all files that have both the same beginning and ending characters. The example selects all files that begin with `t` and end with `file`.	`tst_1_file` `tst_2_file`
`infile=loc('f*.txt')`	An asterisk can represent the `NULL` string.	`f.txt`

The example below shows how to use a wildcard to transfer all files whose filename starts with the letter **`f`** and which have an extension of **`.sas`**. The specified files will be downloaded from the **`/user/progs`** directory on a UNIX server to the **`c:\Users\test`** directory on a Windows client.

See “FILENAME” on page 201

Example

```
filename locHost 'c:\Users\test';
rsubmit;
    filename remHost '/user/progs';
    proc download infile=remHost('f*.sas')
                  outfile=locHost;
    run;
endrsubmit;
```

Example “Example 2: Use a FILENAME Statement with the UPLOAD and DOWNLOAD Procedures ” on page 203.

'external-file-name'
is used to explicitly define the file that is to be uploaded.

```
infile='filename.txt'
```

INLIB=*client-SAS-library*
specifies a SAS library to upload from the client to the server. This option must be used with the OUTLIB= option. Before using this option, you must define the libref that is used for *client-SAS-library*.

Alias IN=, INDD=

MEMTYPE=(*mtype-list*)
specifies one or more member types to be uploaded.

Here are the valid member types:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias MTYPE=, MT=

Requirement To use this option, you must also specify the INLIB= and OUTLIB= options.

OUTCAT=*server-SAS-catalog*
names the SAS catalog that you want to upload to. If you want to create a permanent SAS catalog, you must define the libref before specifying the PROC UPLOAD statement, and you must specify a two-level SAS catalog name. To upload all of the catalogs in a SAS library, specify OUTCAT=*libref*._ALL_.

> **TIP** If you transfer a catalog that contains entries of type PROGRAM, you must compile the entries on the target operating environment before execution. To compile all the PROGRAM entries in a catalog, submit (or remotely submit) the following statements:
>
> ```
> proc build cat=libref.member-name batch;
>     compile;
> run;
> ```
>
> *libref* identifies the SAS library that contains the catalog, and *member-name* identifies the catalog.

Requirement If you use the OUTCAT= option, you must also use the INCAT= option. If you specify the _ALL_ option in OUTCAT=, you must also specify _ALL_ in the INCAT= option.

OUTFILE=*server-file-identifier*
specifies an external file in the server session to which the file in the client session will be transferred.

Here are the values for *server-file-identifier*:

"external-filename"
is the physical location of the file in the server session to which the file in the client session is transferred.

Note: Enclose the filename in double or single quotation marks.

fileref
is the SAS filename that is associated with the physical location of a single file in the server session.

Note: You must define the fileref before you can specify it in the PROC UPLOAD statement.

fileref(member)
is the fileref that is associated with an aggregate storage location, such as a directory or a partitioned data set, in the server session. *member* specifies the file in the aggregate storage location that will be transferred.

Note: If a wildcard (*) is used in the INFILE= option, then OUTFILE=*fileref* should point to an aggregate storage location such as a directory.

Requirement If you use the OUTFILE= option, you must also use the INFILE= option.

OUTLIB=*server-SAS-data-set* <(*SAS-data-set-options*)>
OUT=
specifies the SAS data set in the server session that you want the uploaded data set written to. If you want to create a permanent SAS data set, you must define the libref before specifying the PROC UPLOAD statement, and you must specify a two-level SAS data set name.

The transfer of a long name that might be assigned to a data set is restricted by the SAS release that you are using. SAS releases after SAS 6 support long names assigned to a data set. If a data set that has a long name is transferred to a server that runs SAS 6 or earlier, the long name is truncated. For details about long names, see *SAS Language Reference: Concepts*.

The OUT= option is a valid form of the OUTLIB= option. The UPLOAD procedure determines the meaning of the OUT= option as follows:

- If you specify the DATA= option and the OUT= option, the OUT= option names the output SAS data set.

  For example, if the USER= option is set to MyLib, then the following statement uploads the data set A from the library MyLib on the client to the library MyLib on the server:

```
proc upload data=a out=a;
run;
```

- If you specify only the OUTLIB= option, the UPLOAD procedure uploads the last SAS data set that was created on the client.

  For example, the following statement uploads the last data set that was created on the client to the data set MyData in the library MyLib on the server (assuming USER=MyLib).

```
proc upload out=mydata;
```

```
run;
```

- If you specify the INLIB= option and the OUTLIB= option, the OUTLIB= option specifies the name of a SAS library.

  For example, the following statement uploads all of the data sets and catalogs that are in the library A on the client to the library RmtLib on the server.

```
proc upload inlib=a outlib=rmtlib;
run;
```

For details about the effect of omitting the OUTLIB= option, see "Default Naming Conventions for Uploaded Data Sets" on page 230.

Interaction Most SAS data set options that were used to characterize the data set when it was created will not be inherited when the OUT= option is used. Only the LABEL= and TYPE= data set options are inherited. However, you can explicitly specify SAS data set options as arguments to the OUT= option when uploading a data set. For example, specifying the COMPRESS=YES data set option would cause all observations in the data set to be compressed. You use SAS data set options to change the data set's characteristics or to apply new characteristics.

See "DATA=*client-SAS-data-set* <(*SAS-data-set-options*)>" on page 222

SAS Data Set Options: Reference

Example "Specify Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 231

OUTLIB=*server-SAS-library*
names the destination SAS library on your server where the uploaded data sets and catalogs from the client are stored. Before using this option, you must define the libref that is used for *server-SAS-library*.

Note: The OUTLIB= form of this option is the same as the OUT= option that is used to specify a SAS data set. When you use the OUTLIB= option, the UPLOAD procedure determines whether the input option was DATA= or INLIB= and processes the uploaded objects appropriately.

Alias OUTDD=, OUT=

VIEWTODATA
for a library transfer only, causes view descriptor files to be transferred as data sets instead of as view files, which is the default. If you want some views to be transferred as view files and other views to be transferred as data sets, you would have to perform two separate transfers. If you attempt to use this option for a single data set transfer (by using the DATA= option), an error results.

V6TRANSPORT
specifies that data should be translated by using the SAS 6 "File Format Translation Algorithms" on page 462. Specify this option only when you want to use the SAS 6 translation style explicitly and both the client and the server run SAS 8 or a later release.

When V6TRANSPORT is specified, the default behavior is to promote a numeric variable whose length is less than 8 bytes. To prevent a promotion of this length, you can use the EXTENDSN=NO option along with the V6TRANSPORT option.

XATTR=YES | NO

specifies whether to allow for the upload or download of extended attributes that are defined on a SAS data set or SAS library. This option is turned on by default in PROC UPLOAD and PROC DOWNLOAD. The XATTR=YES option is invalid when the OUT= option is specified.

If XATTR=YES is specified with the OUT= option, then XATTR=YES is ignored and a WARNING is sent to the SAS log. For example, the following statement will cause a WARNING to be sent to the SAS log and no extended attributes will be transferred:

```
proc upload data=inlib.sales out=outlib.sales xattr=y;
run;
```

Extended Attributes are not transferred when the OUT= option is specified with DATA= on PROC DOWNLOAD or PROC UPLOAD. If the XATTR= option is not specified but the DATA= and OUT= options are, then the data set will be transferred, but no extended attributes will be transferred. For example, the following PROC UPLOAD statement will cause the data set Sales to be transferred without its extended attributes:

```
proc download data=inlib.sales out=outlib.sales;
run;
```

If neither the XATTR= nor the OUT= option is specified on PROC UPLOAD or PROC DOWNLOAD then extended attributes will be transferred. For example, the following PROC UPLOAD statement will cause the data set Sales to be uploaded along with its extended attributes:

```
proc upload data=inlib.sales;
run;
```

For information about PROC UPLOAD options and the default behavior of data set options on data sets being transferred, see Table 14.9 on page 232.

Default YES

Restriction If the XATTR=YES and the OUT= option are used together in a PROC UPLOAD statement, then extended attributes defined on the variables in the DATA= data set will not be re-created on the server.

Example “Example 19: Transfer Data Sets with Extended Attributes” on page 249

Details

Default Naming Conventions for Uploaded Data Sets

If you omit the OUT=<*output-data-set*> option, from the UPLOAD statement, SAS follows these rules to determine the name for the data set:

- If the input data set (the data set that is specified in the DATA= option) has a two-level name and the same libref that is defined for the input data set is also defined in the server session, the data set is uploaded to the library on the server that is associated with that libref. The data set has the same member name on the server.

  For example, suppose you submit the following statement:

  ```
  libname orders
      client-SAS-library;
  ```

  If you remotely submit the following statements, the data set Orders.Qtr1 is uploaded to Orders.Qtr1 on the server.

  ```
     /*****************************************/
     /* The libref ORDERS is defined in both    */
     /* operating environments.                 */
     /*****************************************/
  libname orders
     server-SAS-library;
  proc upload data=orders.qtr1;
  run;
  ```

- If the input data set has a two-level name but the libref for the input data set is not also defined in the server session, then the data set is uploaded to the default library on the server. This is usually the Work library, but the library might also be defined by using the User libref.

  The data set retains the same data set name that it had on the client. For example, if you remotely submit the following statement, the data set is uploaded to Work.Qtr2 on the server.

  ```
     /*****************************************/
     /* The libref ORDERS is defined only on    */
     /* the client.                          */
     /*****************************************/
  proc upload data=orders.qtr2;
  run;
  ```

- If the input data set has a one-level name and the default libref on the client also exists on the server, the data set is uploaded to that library.

  For example, suppose you submit the following statements:

  ```
  libname orders
     client-SAS-library;
  options user=orders;
  ```

  If you remotely submit the following statements, the data set Orders.Qtr1 is uploaded to Orders.Qtr1 on the server.

  ```
     /*****************************************/
  ```

```
   /* The libref ORDERS is defined in both    */
   /* operating environments.                 */
   /*******************************************/
libname orders
   server-SAS-library;
libname remote
   server-SAS-library;
   /*********************************/
   /* This option has no effect in     */
   /* this case.                       */
   /*********************************/
options user=remote;
proc upload data=qtr1;
run;
```

- If the input data set has a one-level name and the default libref on the client does not exist on the server, then the data set is uploaded to the default library on the server. That is, the User libref on the server is used only if the User libref on the client does not exist on the server.

  For example, suppose you submit these statements:

  ```
  libname orders
     client-SAS-library;
  options user=orders;
  ```

  When you remotely submit the following statements, the data set Orders.Qtr1 is uploaded to Remote.Qtr1 on the server.

  ```
     /*******************************************/
     /* The libref ORDERS is defined only on    */
     /* the server.                          */
     /*******************************************/
  libname remote
     server-SAS-library;
  options user=remote;
  proc upload data=qtr1;
  run;
  ```

- If you omit the DATA= option, the last data set that was created on the client during the SAS session is uploaded to the server, as follows:

  ```
  proc upload;
  run;
  ```

  The naming conventions on the server follow one of the previously described rules, based on how the last data set was created.

Specify Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD

Restrictions on Using Data Set Options

PROC UPLOAD and PROC DOWNLOAD permit you to specify SAS data set options in the DATA= and OUT= options. However, SAS data set options are not supported when using the INLIB= and OUTLIB= options, even when you upload

only data sets. You can specify SAS data set options only in the DATA= and OUT= options of the PROC UPLOAD statement.

You cannot specify SAS data set options in the INLIB= and OUTLIB= options, even when uploading a single data set. A data set option must be associated with a specific SAS data set.

An uploaded SAS data set inherits characteristics from the selected SAS data set options that are listed in this table under any of these conditions:

- DATA= option is used
- INLIB= and OUTLIB= options are used
- DATA=, INLIB=, and OUTLIB= are not used

Table 14.2 *Default SAS Data Set Options for Data Set Uploads*

SAS Data Set Option	Definition	Inherited When PROC UPLOAD DATA= Is Used	Inherited When PROC UPLOAD OUT= Is Used
ALTER=	Specifies a password for ALTER protection.	Yes	No
COMPRESS	Specifies whether to compress observations, or specifies the compression method.	Yes	No
DROP=	For an input data set, excludes the specified variables from processing; for an output data set, excludes the specified variables from being written to the data set.	Yes	No
GENMAX=	Specifies the maximum number of generations.	Yes	No
INDEX=	Specifies whether to index a data set. The index for an uploaded SAS data set is created on the server, not transferred from the client. To prevent the creation of the index, you can specify the INDEX=NO option in the PROC UPLOAD statement.	Yes	No
KEEP=	For an input data set, specifies the variables to process; for an output data	Yes	No

SAS Data Set Option	Definition	Inherited When PROC UPLOAD DATA= Is Used	Inherited When PROC UPLOAD OUT= Is Used
	set, specifies the variables to write to the data set.		
LABEL=	Specifies whether to label a data set.	Yes	Yes
READ=	Specifies a password for read protection.	Yes	No
RENAME=	Changes the name of a variable.	Yes	No
REUSE=	Specifies whether to reuse free space in compressed data sets.	Yes	No
SORTEDBY=	Specifies the variables by which the data set is sorted.	Yes	No
TYPE=	Specifies the data set type.	Yes	Yes
WRITE=	Specifies the password for WRITE protection.	Yes	No

Examples

Example 1: KEEP= Option

In this example, the KEEP= SAS data set option is used as an argument to the DATA= option in PROC UPLOAD. Because the OUT= option is omitted, the uploaded data set inherits the characteristics of the input data set, including a default action to re-create the index. For details about the KEEP= data set option and a complete list of SAS data set options, see *SAS Data Set Options: Reference*.

```
proc upload data=study(keep=age score1 score2);
run;
```

Example 2: OUT= Option

In this example, because the OUT= option is specified, the uploaded data set does not inherit the characteristics of the input data set **`study`**. Instead, the data set is renamed as **`results`** in the server session. The uploaded data set also inherits only the LABEL= and TYPE= data set options. For details about the LABEL= and TYPE= SAS data set options, see *SAS Data Set Options: Reference*.

```
proc upload data=study  out=results;
run;
```

Example 3: KEEP= and OUT= Options

In this example, the KEEP= SAS data set option is used as an argument to the OUT= option in PROC UPLOAD. Because the OUT= option is specified, the uploaded data set does not inherit the characteristics of the input data set **study**. Instead, the data set is renamed as **results** in the server session. The uploaded data set also inherits only the LABEL= and TYPE= data set options. The INDEX=NO data set option specifies that the index will not be re-created in the server session.

For details about the LABEL=, TYPE=, and KEEP= SAS system options, see *SAS Data Set Options: Reference*.

```
proc upload data=study  out=results(keep=age score1 score2) index=no;
run;
```

WHERE Statement

Selects observations from SAS data sets.

Restriction: The UPLOAD procedure processes WHERE statements when you transfer a single SAS data set.

See: *SAS Data Set Options: Reference*.

Syntax

WHERE *where-expression-1 <logical-operator where-expression-n>*;

Syntax Description

where-expression-1
is a WHERE expression.

logical-operator
is one of the following logical operators:

- AND
- AND NOT
- OR
- OR NOT

where-expression-n
is a WHERE expression.

WHERE statements allow multiple WHERE expressions that are joined by logical operators.

You can use SAS functions in a WHERE expression. Also, note that a DATA step or a PROC step attempts to use an available index to optimize the selection of data when an indexed variable is used in combination with one of the following:

- CONTAINS operator
- LIKE operator
- colon modifier with a comparison operator
- TRIM function
- SUBSTR function (in some cases)

To understand when using the SUBSTR function causes an index to be used, look at the format of the SUBSTR function in a WHERE statement:

```
where substr(variable, position, length)
   ='character-string';
```

An index is used in processing when all of the following conditions are met:

- position is equal to 1
- length is less than or equal to the length of variable
- length is equal to the length of character-string

The following example illustrates using a WHERE statement with the UPLOAD procedure. The uploaded data set contains only the observations that meet the WHERE condition.

```
proc upload data=revenue out=new;
   where origin='Atlanta' and revenue < 10000;
run;
```

For details, see the *SAS Data Set Options: Reference*.

EXCLUDE Statement

Excludes library members or catalog entries from uploading.

Restriction: You cannot use the EXCLUDE and SELECT statements in the same PROC UPLOAD step.

Syntax

EXCLUDE *lib-member-list* </ MEMTYPE=*mtype* >;
EXCLUDE *cat-entry-list* </ ENTRYTYPE=*etype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= options in the PROC UPLOAD statement. Use the format *cat-entry-list* </

ENTRYTYPE=*etype*> when you specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

lib-member-list
specifies which library members to exclude from uploading. You can name each member explicitly or use one of the following forms:

prefix
specifies all members whose names begin with the character string *prefix*. For example, if you specify `TEST:`, all members with names that begin with the letters `TEST` are excluded.

first-last
specifies all members whose names have a value between *first* and *last*. For example, if you specify `TEST1-TEST3`, any files that are named `TEST1`, `TEST2`, or `TEST3` are excluded.

Restriction *first* and *last* must begin with identical character strings and must end in a number.

cat-entry-list
specifies which catalog entries to exclude from uploading. Each element of *cat-entry-list* has the form *entry*.*type*.

entry
is the name of an entry in the *catalog* to exclude from uploading.

.type
is the type of the catalog entry. This part of the name is optional.

MEMTYPE=*mtype*
specifies a member type to exclude from uploading.

Here are the valid member types:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias MTYPE=, MT=

Requirement To use this option, you must also specify the INLIB= and OUTLIB= options in the PROC UPLOAD statement.

ENTRYTYPE=*etype*
specifies a catalog entry type to exclude from uploading. Examples of catalog entry types include FORMAT and DATA.

Alias ETYPE=, ET=

Requirement To use this option, you must specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

SELECT Statement

Selects specific library members or catalog entries to upload.

Restriction: You cannot use the EXCLUDE and SELECT statements in the same PROC UPLOAD step.

Syntax

SELECT *lib-member-list* </ MEMTYPE=*mtype*>;
SELECT *cat-entry-list* </ ENTRYTYPE=*etype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= options in the PROC UPLOAD statement. Use the format *cat-entry-list* </ ENTRYTYPE=*etype*> when you specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

lib-member-list
specifies which library members to exclude from uploading. You can name each member explicitly or use one of the following forms:

prefix
specifies all members whose names begin with the character string *prefix*. For example, if you specify `TEST:`, all members with names that begin with the letters `TEST` are excluded.

first-last
specifies all members whose names have a value between *first* and *last*. For example, if you specify `TEST1-TEST3`, any files that are named `TEST1`, `TEST2`, or `TEST3` are excluded.

Restriction *first* and *last* must begin with identical character strings and must end in a number.

cat-entry-list
specifies which catalog entries to exclude from uploading. Each element of *cat-entry-list* has the form *entry*.*type*.

entry
is the name of an entry in the *catalog* to exclude from uploading.

.type
is the type of the catalog entry. This part of the name is optional.

MEMTYPE=*mtype*
specifies a member type to exclude from uploading.

Here are the valid member types:

- ALL

- CATALOG
- DATA
- MDDB
- VIEW

Alias MTYPE=, MT=

Requirement To use this option, you must also specify the INLIB= and OUTLIB= options in the PROC UPLOAD statement.

ENTRYTYPE=*etype*
specifies a catalog entry type to exclude from uploading. Examples of catalog entry types include FORMAT and DATA.

Note: The SELECT statement also enables you to maintain an ordering and grouping of catalog entries that contain graphics output because entries are uploaded into the server SAS catalog on the order in which you specify them in the SELECT statement.

Alias ETYPE=, ET=

Requirement To use this option, you must specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

TRANTAB Statement

Specifies the translation table to use when translating character data for an upload from a SAS/CONNECT client to a SAS/CONNECT server.

Restriction: You can specify only one translation table per TRANTAB statement. To specify additional translation tables, use additional TRANTAB statements.

Requirement: To use the TRANTAB statement, you must specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

See: *SAS Data Set Options: Reference*.

Syntax

TRANTAB NAME=*translation-table-name*
<*options*>;

Usage: UPLOAD Procedure

Using: UPLOAD Procedure

VALIDMEMNAME and VALIDVARNAME System Options

If the data that you are transferring contains an invalid SAS name, such as a name containing special characters, national characters, or embedded blanks, then you can specify VALIDVARNAME=ANY or VALIDMEMNAME=EXTEND before the sign-on statement to successfully transfer the files. The following types of data can contain nonstandard SAS names when you use the VALIDVARNAME and VALIDMEMNAME system options with PROCS UPLOAD and DOWNLOAD:

- a SAS data set
- a SAS library
- a SAS variable
- a DBMS table
- a DBMS table column heading

Note: You must specify the VALIDMEMNAME and VALIDVARNAME system options before the SIGNON statement.

For more information about these Base SAS system options, see *SAS System Options: Reference*.

Results: UPLOAD Procedure

Results: UPLOAD Procedure

The UPLOAD procedure writes a series of informative messages to the SAS log when it executes. Examples of these messages are shown in this output:

Output 14.1 *SAS Log Messages from the UPLOAD Procedure*

```
NOTE: Remote submit to B commencing.
 1    proc upload infile='client-external-file'
 2       outfile='server-external-file';run;

 NOTE: TEXT upload in progress from infile=client-external-file
       to outfile=server-external-file
 NOTE: Uploaded 4 records and 136 bytes.
 NOTE: 4 records were read from the file client-external-file
       The maximum record length was 65.
       The minimum record length was 0.
 NOTE: 136 bytes were transferred at 68 bytes/second.
 NOTE: The PROCEDURE UPLOAD used 0.04 CPU seconds and 1431K.

 NOTE: Remote submit to B complete.
$
```

Examples: UPLOAD Procedure

Example 1: Transfer Specific Member Types

If you specify the INLIB= and OUTLIB= options in the PROC UPLOAD or PROC DOWNLOAD statements, you can specify which member types to transfer by using the MEMTYPE= option in one of the following statements:

- PROC UPLOAD
- PROC DOWNLOAD
- SELECT

- EXCLUDE

Valid values for the MEMTYPE= option are DATA, CATALOG, MDDB view, FDB, and ALL. If you use this option in the SELECT or EXCLUDE statement, you can specify only one value. If you use this option in the PROC UPLOAD or the PROC DOWNLOAD statement, you can specify a list of MEMTYPE values enclosed in parentheses. This example uploads all data sets and catalogs that are in the library *This* on the client and stores them in the library *That* on the server.

```
rsubmit;
  proc upload inlib=this outlib=that
  memtype=(data catalog);
endrsubmit;
```

Example 2: The MEMTYPE= Option in the PROC UPLOAD Statement

This example uploads all catalogs and data sets that are in the library Loclib on the client, except the data sets that are named Z4, Z5, Z6, and Z7. It then stores them in the library Remlib on the server:

```
rsubmit;
  proc upload inlib=loclib outlib=remlib mt=all;
    exclude z4-z7 / memtype=data;
  run;
endrsubmit;
```

Example 3: Transfer Specific Catalog Entry Types

When you include the INCAT= and OUTCAT= options in the PROC UPLOAD or PROC DOWNLOAD statement, you can specify which entry types to transfer by using the ENTRYTYPE= option in one of the following statements:

- PROC UPLOAD
- PROC DOWNLOAD
- SELECT
- EXCLUDE

This example uploads all Slist catalog entries from the Cat catalog in the library Loclib on the client and stores them in the catalog Upcat in the library Remlib on the server:

```
rsubmit;
  proc upload incat=loclib.cat
    outcat=remlib.upcat entrytype=slist;
  run;
endrsubmit;
```

Example 4: The ENTRYTYPE= Option in the SELECT Statement in PROC UPLOAD

If the default library is Work, this example uploads the FORMAT catalog entries XYZ and ABC, the INFMT catalog entry Grades, and the SCL entries A and B that are in the Work.Locfmt catalog on the client. It then stores them in the Work.Remfmt catalog on the server: If you omit the ENTRYTYPE= option and also omit the SELECT and EXCLUDE statements, all catalog entries are transferred.

```
proc format lib=work.locfmt;
  invalue grades 'one'=1;
  value abc 1='one';
  value xyz 1='one';
run;
rsubmit;
  proc upload incat=locfmt outcat=remfmt;
    select xyz.format grades
      abc (et=format) / et=infmt;
    select a b / et=scl;
  run;
endrsubmit;
```

Example 5: Long Member Names in Catalog Transfers

This example uses PROC UPLOAD to transfer entire catalogs by using both the INCAT= and OUTCAT= options:

```
rsubmit;
  proc upload
    incat=loclib.monthlysalary
    outcat=monthlyupdate;
  run;
  proc upload
    incat=loclib.employeedata
    outcat=remlib.cat;
  run;

  proc upload incat=sasuser.base
    outcat = remlib.basecatalog;
  run;

endrsubmit;
```

Example 6: Use LIBRARY Transfers to Transfer Data Set Generations

Generation data sets are historical versions of SAS data sets, SAS views, and SAS/ACCESS files. They enable you to keep a historical record of the changes that you make to these files. There are two data set options that are useful when manipulating generations of SAS data sets: GENMAX (maximum number of generations) and GENNUM (generation number). GENMAX specifies how many generations to keep, and GENNUM is used to access a specific version of a generation group. SAS/CONNECT transfers generations of SAS data sets by default during library transfers. The base data set, as well as all of its historical versions, are transferred. If you do not want all generations to be transferred, you should do one of the following:

- transfer a library using the GEN=NO option.
- transfer single data sets. Only the specified data set is transferred.

This example transfers the client data set Local.Sales as well as its generations to the server library Remote. If the data set Sales already exists in the output library, the base and all existing generations are deleted and replaced by those that are uploaded.

```
data local.sales(genmax=3);
   input store sales95 sales96 sales97;
   datalines;
  1   221325.85   214664.02   212644.60
  2   134511.96   159369.47   317808.48
  3   321662.42   244789.33   236782.59
 ;
run;

data local.sales;
   input store sales95 sales96 sales97;
   datalines;
  1   251325.25   217662.16   222614.60
  2   144512.11   179369.47   327808.48
  3   329682.43   249989.93   256782.59
 ;
run;

data local.sales;
   input store sales95 sales96 sales97;
   datalines;
  1   261325.33   218862.16   222614.60
  2   145012.11   189339.47   328708.71
  3   330682.46   259919.92   258722.52
 ;
run;

   /* PROC DATASETS will show that the   */
   /* base data set as well as two       */
   /* generations exist in the library.  */
```

```
proc datasets lib=local;
quit;

rsubmit;
   proc upload in=local out=remote cstatus=no;
   run;
endrsubmit;
```

Example 7: Use a SELECT Statement to Transfer Generations

Specific generations of data sets cannot be specified in the SELECT or the EXCLUDE statements for library transfers. When the SELECT statement is specified for the library transfer, the selected base data set as well as all of its historical versions are transferred. Similarly, when the EXCLUDE statement is specified for the library transfer and the GEN=NO option is not specified, the selected base data set as well as all of its historical versions are excluded from the transfer.

In the following example, the data set Local.Sales as well as all of its generations are uploaded.

```
data local.sales(genmax=3); x=1; run;
data local.sales; x=2; run;
data local.sales ; x=3; run;
data local.x; x=1; run;
rsubmit;
  proc upload in=local out=remote;
    select sales (mt=data);
  run;
endrsubmit;
```

Example 8: Transfer Single Data Sets Using PROC UPLOAD

A specific generation of data set can be transferred by specifying the GENNUM= data set option for a single data set transfer. In the following example, a specific historical version is uploaded by specifying GENNUM=1.

```
rsubmit;
  proc upload data=local.sales(gennum=1);
  run;
endrsubmit;
```

Example 9: The DROP= Option in the PROC UPLOAD Statement

This example uploads the SAS data set *Loc* in the library Work on the client to the library Work on the server. The variable *One* is dropped from the output data set. Any non-referential integrity constraints that are defined for the input data set that do not include the variable *One* are inherited by the output data set.

```
rsubmit;
  proc upload data=loc(drop=one);
  run;
endrsubmit;
```

Example 10: The INLIB= Option in the PROC UPLOAD Statement

This example uploads all SAS data sets in the library Sasuser on the client and stores them in the library Work on the server. Any non-referential integrity constraints that are defined for each of the input data sets are inherited by the corresponding output data set.

```
rsubmit;
  proc upload inlib=sasuser outlib=work;
  run;
endrsubmit;
```

Example 11: The EXTENDSN= and V6TRANSPORT Options in the PROC UPLOAD Statement

For SAS releases prior to SAS 8, when you transfer short numerics (length less than 8), the length of these numerics is automatically increased to preserve precision. In SAS 8, the length of these numerics is not increased by default unless the V6TRANSPORT option is specified. Using the V6TRANSPORT and EXTENDSN= options in PROC UPLOAD and PROC DOWNLOAD statements, you can choose whether to promote the length of numerics.

This example uploads the data set A in the directory Work on the client to the directory Remote on the server. The V6TRANSPORT option causes the short numerics to be promoted. Therefore, EXTENDSN=NO must be specified to override this default, so that numerics will not be promoted.

```
rsubmit;
  proc upload data=a out=remote
```

```
    v6transport extendsn=no;
  run;
endrsubmit;
```

Example 12: Transfer SAS Utility Files

You can use the INLIB= and OUTLIB= options with PROC UPLOAD or PROC DOWNLOAD to transfer multiple SAS files in a single step. This capability enables you to transfer an entire library or selected members of a library. Make sure that the INLIB= option must be used with the OUTLIB= option. You can specify which member types to transfer by using the MEMTYPE= option in one of the following statements: PROC UPLOAD, PROC DOWNLOAD, SELECT, and EXCLUDE. If you use the MEMTYPE= option in the SELECT or the EXCLUDE statement, you can specify only one value. If you use the MEMTYPE= option in the PROC UPLOAD or the PROC DOWNLOAD statement, you can specify a list of MEMTYPE values enclosed in parenthesis. Here are the valid values for the MEMTYPE= option:

- DATA (SAS data sets)
- CATALOG (SAS catalogs)
- VIEW (SQL views)
- MDDB (MDDB files)
- ALL (all of the preceding values)

This example uploads all SAS data sets, catalog files, SQL views, and MDDB files in the library Work on the server and stores them in the library Work on the client:

```
rsubmit;
  proc upload inlib=work outlib=work;
  run;
endrsubmit;
```

Example 13: The MEMTYPE= Option in the PROC UPLOAD Statement

This example uploads all MDDB and FDB files that are in the library. This on the client and stores them in the library That on the server:

```
rsubmit;
  proc upload inlib=this outlib=that
    memtype=(mddb view);
  run;
endrsubmit;
```

Example 14: The MEMTYPE= Option in the SELECT Statement

This example uploads the MDDB files Test1 and Test2 and the SAS data set Test3 that are in the library Work on the server and stores them in the library Local on the client:

```
rsubmit;
  proc upload inlib=work outlib=local;
    select test1 test2 test3(mt=data)/memtype=mddb;
  run;
endrsubmit;
```

Example 15: The MEMTYPE= Option in the EXCLUDE Statement

This example uploads all SAS data sets, catalog files, MDDB files, FDB files, and SQL views that are in the library Local on the client, except the SQL views A1, A2, A3. If then stores them in the library Remote on the server:

```
rsubmit;
        proc upload inlib=local outlib=remote memtype=all;
      exclude a1-a3/memtype=view;
    run;
endrsubmit;
```

Example 16: Distribute an .EXE File from the Server to Multiple Clients: UPLOAD

SAS/CONNECT facilitates the distribution of information to multiple clients. Rather than distributing files on CD-ROMs, you can make one central file available on the server that each client can access and copy. For example, suppose that you want to distribute an updated executable to other Windows computers in your organization. You decide that the most efficient way to update all computers is to upload PROGRAM.EXE to the server, and notify each person who uses this software on their workstations that the file is available and should be downloaded. This method enables all clients to quickly access the updated software, and eliminates the need to share a physical CD-ROM among client users.

Note: Such a SAS/CONNECT application, in which an external nontext file is uploaded and then downloaded, requires the BINARY option in the DOWNLOAD and UPLOAD procedures. The BINARY option transfers files without any character translation (for example, EBCDIC to ASCII) or insertion of record delimiters.

The INFILE= and OUTFILE= options are specified in the PROC UPLOAD statement in order to upload an external file. To upload a SAS data set, the DATA= and OUT= options should be used.

The PROGRAM.DLL module must first be uploaded to an external file on the server. This example uses a SAS FILENAME statement to identify the target file on the server.

```
rsubmit;
   filename rfile 'server-file';
   proc upload infile='a:\program.dll'
      outfile=rfile binary;
   run;
endrsubmit;
```

Example 17: Distribute an .EXE File from the Server to Multiple Clients: DOWNLOAD

With the PROGRAM.DLL module available on the server, each client at the installation can acquire the updated module by downloading it from the server.

The process for downloading the PROGRAM.DLL module is like the process for uploading, except that the DOWNLOAD procedure is invoked, and the target file is on the server, not on the client. The following example copies the PROGRAM.DLL module to directory \SAS\SASEXE.

This example uses a SAS FILENAME statement to identify the target file on the server. The INFILE= and OUTFILE= options are used in the PROC DOWNLOAD statement.

```
rsubmit;
   filename rfile 'server-file';
   proc download infile=rfile
      outfile='\sas\sasexe\program.dll' binary;
   run;
endrsubmit;
```

Example 18: Create an Index with OUT= Using PROC UPLOAD

The purpose of the INDEX=**YES**|**NO** procedure option is to preserve indexes on data sets that are being transferred to a server session during a PROC UPLOAD. When OUT= is specified, the indexes are not transferred.

In this example, the INDEX=YES option specifies that an index will be re-created in the server session. However, because OUT= is specified, the indexes defined on the DATA= data set will not be created and a WARNING will be issued in the log. The INDEX=*Region* data set option causes an index file to be created and associated with the DATA set *Sales* in the server session. The index file identifies all the observations that contain the variable *Region* and its associated values.

```
rsubmit;
  proc upload index=yes data=sales out=sales(index=(region));
  run;
endrsubmit;

proc contents data=sales;
run;
```

Output 14.2 *Partial PROC CONTENTS Output for INDEX= and OUT= with PROC UPLOAD*

```
Alphabetic List of Indexes and Attributes

                          # of
                        Unique
          #    Index    Values

          1    region        3
```

Example 19: Transfer Data Sets with Extended Attributes

In the following example, the extended attributes will not be transferred because the OUT= option is specified. The variable Purchase will be successfully dropped.

```
signon;
rsubmit;
  libname inlib "/path/to/inlib";
endrsubmit;
rsubmit;
  libname outlib "/path/to/outlib";
endrsubmit;

   data inlib.sales;
      purchase = "car";
      age = 10;
      income = 20000;
      kids = 3;
      cars = 4;
    run;
    /* Create the Extended Attributes */
   proc datasets lib=inlib nolist;
      modify sales;
         /* changing from the default of 200 */
         xattr options maxchunk=100;
         xattr add ds role="train" attrib="table" numAttribute=12345;
         xattr add var purchase ( role="target" level="nominal" )
                       age      ( role="reject"
            numAttribute1=12345678901234567890123456789012345678901
            numAttribute2=-12345678901234567890123456789012345678901 )
```

```
                        income    ( role="input" level="interval" );
   run;
quit;
rsubmit;
   proc upload data=inlib.sales out=outlib.sales(drop=purchase);
   run;
endrsubmit;
```

Example 20: Compute Services and Data Transfer Services Combined: Macro Capabilities

Regardless of the motivation for reducing the amount of data that is transferred, incorporating Compute Services will achieve your goal. Compute Services enables you to format and pre-process data into a subset or a summarized form in the server session before transferring the subsequent smaller amount of data to the client session. This balances the use of CPU cycles between the client and server sessions and minimizes the amount of data contributing to network traffic. SAS/CONNECT is fully functional from within the macro facility. Both the UPLOAD and the DOWNLOAD procedures can update the macro variable SYSINFO and set it to a nonzero value if the procedure terminates because of errors. You can also use the %SYSRPUT macro statement in the server session to send the value of the SYSINFO macro variable back to the client session. Thus, you can submit a job to the server and test whether a PROC UPLOAD or a PROC DOWNLOAD step successfully completed before beginning another step in either the client or server session.

This program includes a transaction file that is located on the client, which will be uploaded to a server in order to update a master file. You can test the results of the PROC UPLOAD step in the server session by checking the value of the SYSINFO macro variable. The SYSINFO macro variable can be used to determine whether the transaction file was successfully uploaded. If successful, the master file is updated with the new information. If the upload was not successful, you receive a message that explains the problem. You can use the %SYSRPUT macro statement to send the return code from the server session back to the client session. The client session can test the results of the upload and, if it is successful, use the DATASETS procedure to archive the transaction data set.

```
1 libname trans
'client-SAS-library';
   libname backup
'client-SAS-library';2
 rsubmit;3
    proc upload data=trans.current out=current;
      run;
4
    %sysrput upload_rc=&sysinfo;
      %macro update_employee;
5
       %if &sysinfo=0 %then %do;
            libname perm
'server-SAS-library';
            data perm.employee;
               update perm.employee current;
```

```
                by employee_id;
              run;
            %end;
6
         %else %put ERROR: UPLOAD of CURRENT
                      failed. Master file was
                      not updated.;
       %mend update_employee;7
     %update_employee;
   endrsubmit;
8
 %macro check_upload;9
    %if &upload_rc=0 %then %do;10
       proc datasets lib=trans;
            copy out=backup;
         run;
      %end;
   %mend check_upload;11
 %check_upload;
```

1 Associates a libref with the SAS library that contains the transaction data set and backup data in the client session.

2 Sends the PROC UPLOAD statement and the UPDATE_EMPLOYEE macro to the server session for execution.

3 Because a single-level name for the OUT= argument is specified, the PROC UPLOAD step stores CURRENT in the default library (usually Work) in the server session.

4 If the PROC UPLOAD step successfully completes, the SYSINFO macro variable is set to 0. The %SYSRPUT macro statement creates the UPLOAD_RC macro variable in the client session, and puts the value that is stored in the SYSINFO macro variable into UPLOAD_RC. The UPLOAD_RC macro variable is passed to the client session and can be tested to determine whether the PROC UPLOAD step was successful.

5 Tests the SYSINFO macro variable in the server session. If the PROC UPLOAD step is successful, the transaction data set is used to update the master data set.

6 If the SYSINFO macro variable is not set to 0, the PROC UPLOAD step has failed, and the server session sends messages to the SAS log (which appear in the client session) notifying you that the step has failed.

7 Executes the UPDATE_EMPLOYEE macro in the server session.

8 The CHECK_UPLOAD macro is defined in the client session because it follows the ENDRSUBMIT statement.

9 Tests the value of the UPLOAD_RC macro variable that was created by the %SYSRPUT macro statement in the server session to determine whether the PROC UPLOAD step was successful.

10 When the transaction data set has been successfully uploaded and added to the master data set, the transaction file can be archived in the client session by using the COPY statement in the DATASETS procedure.

11 Executes the CHECK_UPLOAD macro in the client session.

Example 21: RLS and UPLOAD/DOWNLOAD Combined: Distribution of Reports over a Network

When the amount of information that is needed from a server is small (for example, the value of one variable for 12 records or less), Remote Library Services (RLS) can be used to move the data to the client session. When the data is located at the client, the data can be used in a larger processing task, and the results (for example, reports) can be transferred by using PROC UPLOAD across the network as required.

This SCL program fragment enables the distribution of production reports from a company's headquarters location to each of its franchise offices, based on the information that is contained in the control data set that is maintained by each of the franchise offices. This application was implemented by using the macro facility to enable the mainframe to connect with each of the franchise workstations, and to transfer a set of reports to the franchise offices based on selection criteria.

```
      /***********************************/
      /* Name: DISTREPORT.SCL            */
      /*                                 */
      /* This program distributes reports */
      /* to the franchise offices.       */
      /***********************************/
   length rc 8;

  submit continue;
      /***********************************/
      /* set up distribution macro       */
      /***********************************/
 %macro distribution; 1
2
 %let franchise_city=Atlanta NYC LA Dallas Chicago;
   %let franchise_host=
      tsoatl unixnyc unixla wntdal cmshq;
3
 %let j=1;
      %do %while(%scan(&franchise_city,&j) ne );
         %let nextfran=%scan(&franchise_city,&j);
         %let nextrem=%scan(&franchise_host,&j);
         %let j=%eval(&j+1);
options remote=&nextrem  4

comamid=communication-access-method;
   filename rlink 'script-file-name';
   signon;
5
 x "alloc fi(xferrpt)
      da('sasinfo.sugi18.xferrpt') shr";
6
 rsubmit;
      filename frptlib
         "d:\counter\reports\prod";
```

```
   endrsubmit;

      /************************************/
      /* use SAS/CONNECT server           */
      /************************************/
 libname rpt "d:\counter\reports"  7
server=&nextrem; 8
 data _null_;
      set rpt.preport end=finish;
      file xferrpt;
      if _n_ =1 then put "rsubmit;";

         /*********************************/
         /* transfer reports              */
         /* named by variable name in     */
         /* reports data set              */
         /*********************************/

    if (copy="Y") then do; 9
         put "proc upload infile=
            'sasinfo.sugi18."name"'";
         put "outfile=frptlib("name")
         status=no;run;";
      end;
      if finish then put "endrsubmit;";
   run;

      /************************************/
      /* upload reports that you want     */
      /************************************/
 %include xferrpt; 10

   signoff;
   %end;

   %mend;

      /************************************/
      /* invoke macro to distribute       */
      /* reports                          */
      /************************************/
 %distribution; 11
  endsubmit;

   _status_='H';

   return;
```

1 Declares the distribution macro definition.

2 Initializes the list of remote franchise offices (`franchise_city`) and their node names (`franchise_host`) to be used as the REMOTE= value.

3 Scans to the next office and node name to be processed.

4 Specifies the remote office NODENAME as the REMOTE= value and sign on to the remote franchise.

5 Allocates a z/OS file that will contain generated UPLOAD statements.

6 Remotely submits a fileref to define the PC library to which reports will be uploaded.

7 Connects to a server to access the library that contains the report-selection data set.

8 Executes the DATA step to evaluate report-selection data (RPT.PREPORT) and creates UPLOAD statements to transfer reports (XFERRPT).

9 If the selection criterion is YES, creates the appropriate PROC UPLOAD statement for the specified report.

10 Includes the generated SAS job in the client session for execution.

11 Invokes the macro.

Chapter 15

DOWNLOAD Procedure

Overview: DOWNLOAD Procedure

Introduction

After you have started SAS/CONNECT, you can transfer SAS files between your client session and the server. The DOWNLOAD procedure copies SAS files that are stored on the server to the client.

Using PROC DOWNLOAD, you can do the following:

- transfer multiple SAS files in a single step by using the INLIB= and OUTLIB= options. This capability enables you to transfer an entire library or selected members of a library in a single PROC DOWNLOAD step.
- download specific entries in a catalog or specific members in a library by using the SELECT and EXCLUDE statements.
- use WHERE processing and SAS data set options when downloading individual SAS data sets.
- replicate selected data set attributes when downloading a data set.
- transfer data sets and catalog entries that have been modified on or after the specified date.
- specify the translation table to be used when you download a SAS catalog.

See Chapter 5, “Using Data Transfer Services,” on page 87 for information about data transfer services in SAS/CONNECT.

Syntax: DOWNLOAD Procedure

PROC DOWNLOAD
<*data-set-options*>
<*catalog-options*>
<*library-options*>
<*external-file-options*>
<AFTER=*date*>
<CONNECTSTATUS=YES | NO>;

WHERE *where-expression-1* <*logical-operator where-expression-n*>;

EXCLUDE *list* </MEMTYPE=*mtype* | ENTRYTYPE=*etype*>;

SELECT </MEMTYPE=*mtype* | ENTRYTYPE=*etype*>;

TRANTAB NAME=*translation-table-name* <TYPE=(*etype-list)*>
<OPT=DISP | SRC>;

PROC DOWNLOAD Statement

Transfers files from the server to the client.

Alias: none

Syntax

PROC DOWNLOAD

<*data-set-options*>
<*catalog-options*>
<*library-options*>
<*external-file-options*>
<AFTER=*date*>
<CONNECTSTATUS=YES | NO>;

Data Set Options

CAUTION

Do not confuse the PROC DOWNLOAD data set options with the SAS data set options. The PROC DOWNLOAD data set options are valid only in the context of PROC DOWNLOAD. However, two of the PROC DOWNLOAD data set options (DATA= and OUT=) can be further characterized by SAS data set options. For details, see the descriptions for the DATA= on page 222 option and the OUT= on page 227 options.

data-set-options can be one or more of the following:

- "CONSTRAINT=YES | NO" on page 259
- "DATA=*server-SAS-data-set* <(*SAS-data-set-options*)>" on page 260
- "DATECOPY" on page 260
- "EXTENDSN=YES | NO" on page 260
- "INDEX=YES | NO" on page 261
- "OUT=*client-SAS-data-set* <(*SAS-data-set-options*)>" on page 264
- "V6TRANSPORT" on page 266
- "XATTR=YES | NO" on page 266

Catalog Options

catalog-options can be one or more of the following:

- "ENTRYTYPE=*etype* " on page 260
- "EXTENDSN=YES | NO" on page 260
- "INCAT=*server-SAS-catalog*" on page 261

- "OUTCAT=*client-SAS-catalog*" on page 264

Library Options

library-options can be one or more of the following:

- "CONSTRAINT=YES | NO" on page 259
- "EXTENDSN=YES | NO" on page 260
- "GEN=YES | NO" on page 261
- "INDEX=YES | NO" on page 261
- "INLIB=*server-SAS-library*" on page 263
- "MEMTYPE=(*mtype-list*) " on page 263
- "OUTLIB=*client-SAS-library* " on page 265
- "VIEWTODATA" on page 266
- "V6TRANSPORT" on page 266

External File Options

external-file-options are the following:

- "BINARY" on page 259
- "INFILE=*server-file-identifier*" on page 262
- "OUTFILE=*client-file-identifier*" on page 265

Optional Arguments

AFTER=*date*
specifies a modification date in the form of a numeric date value or a SAS date constant.

This option is valid for transferring data sets, catalogs, and libraries. Its use results in data sets or catalog entries being transferred only if they have been modified on or after the specified date.

The AFTER= option is also valid for external file transfers between most computers. If a computer is unable to perform the transfer, this message is displayed:

```
ERROR: AFTER= not supported on this platform.
NOTE: The SAS System stopped processing this step
       because of errors.
```

Note: The AFTER= option is available in SAS 6.09E, SAS 6.11, TS040, and later.

For example, the following statements cause the transfer of data sets only if they were modified within the last week.

```
/************************************/
/* Download all data sets that have */
```

```
    /* been modified in the last week.  */
    /**********************************/
rsubmit;
    data _null_;
    today=date();
    lastweek=today-7;
    call symput('lastweek',lastweek);
    run;
    proc download in=perm out=work
       after=&lastweek memtype=data;
    run;
endrsubmit;
```

If your client session is using an earlier release of SAS that does not support the AFTER= option, then PROC DOWNLOAD still executes this option because the server has the input data set.

BINARY
specifies a download of a binary image (an exact copy) of an external server file. Use this option only for downloading external files.

Note: External files are files that are not SAS files.

By default, if the client and server run in different operating environments (for example, UNIX and Windows), then PROC DOWNLOAD transfers a file from the client to the server, translating the file from UNIX representation to Windows representation. PROC DOWNLOAD also inserts record delimiters that are appropriate for the target environment.

You do not always want to translate a file. For example, you might need to download executable files from the server to the client and later upload them back to the server. Binary file format also saves resources for users who store their own files and for system backups. The BINARY option prevents delimiters from being inserted for each file record that is created at the client. In addition, if the client and server use a different method of data representation, the BINARY option prevents any data translation between ASCII and EBCDIC.

For an example of using the BINARY option, see “Example 16: Distribute an .EXE File from the Server to Multiple Clients: UPLOAD” on page 247.

CONNECTSTATUS=YES | NO
specifies whether the Transfer Status window should be displayed during a transfer. By default, the DOWNLOAD procedure displays the “Transfer Status Window” on page 92.

Alias CSTATUS=, STATUS=

Default YES

CONSTRAINT=YES | NO
specifies if integrity constraints should be re-created on the client when a SAS data set that has integrity constraints defined is downloaded. You can specify this option with the DATA= option (if you omit the OUT= option) or with the INLIB= and OUTLIB= options.

By default, integrity constraints are re-created only when you download a SAS library or when you download a single SAS data set and omit the OUT= option. If you specify the OUT= option with the DATA= option, the integrity constraints are not re-created.

DATA=*server-SAS-data-set* <(*SAS-data-set-options*)>
specifies a SAS data set that you want to download from the server to the client. If the data set is a permanent SAS data set, you must define a libref before the PROC DOWNLOAD statement and specify the two-level name of the data set.

If you specify the name of a data view in the DATA= option, the materialized data is downloaded to the client, not to the view definition.

If you do not specify the DATA=, INCAT=, INFILE=, or INLIB= option, the last SAS data set that was created on the server during your SAS session is downloaded.

Requirement If you specify the DATA= option, you must either use the OUT= option or omit all other options.

See "Specify Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD" on page 231

SAS Data Set Options: Reference

"OUT=*client-SAS-data-set* <(*SAS-data-set-options*)>" on page 264

DATECOPY
retains the date on which a SAS data set was created and the date on which a SAS data set was last modified for each data set that is transferred.

ENTRYTYPE=*etype*
specifies a catalog entry type to be downloaded. Examples of catalog entry types include DATA and FORMAT.

Alias ETYPE=, ET=

Requirement To use this option, you must also specify the INCAT= and OUTCAT= options.

EXTENDSN=YES | NO
specifies whether to promote the length of short numerics (length less than 8 bytes) when transferring.

NO
indicates that the length of numeric variables is not promoted.

YES
indicates that 1 will be added to the length of any numeric variable that has a length of less than 8 bytes before it is transferred to the client computer.

The behavior of the EXTENDSN= option varies according to the SAS release that is used.

- If both the client and the server run SAS 8 or a later release, and the V6TRANSPORT option is specified, then the default is to promote the length of the numeric variable whose length is less than 8 bytes. This is consistent with SAS 6 behavior. To override this behavior, specify EXTENDSN=NO along with the V6TRANSPORT option in the DOWNLOAD statement.
- If either the client or the server runs SAS 6, neither the V6TRANSPORT nor the EXTENDSN= option is supported or recognized.
- If the client runs SAS 6 and the server runs SAS 8 or a later release, a numeric variable whose length is less than 8 bytes is promoted by default. In

this case, specify EXTENDSN=NO in order to override the SAS 6 default and to prevent the promotion.

See "File Format Translation Algorithms" on page 462 for information about translating file formats between a client and server that run on computers whose internal representations are incompatible.

Default NO

GEN=YES | NO
specifies that data set generations are to be sent during library transfers.

YES
specifies that data set generations are sent during library transfers.

NO
specifies that data set generations are not sent during library transfers.

Default YES

INCAT=*server-SAS-catalog*
names a SAS catalog that you want to download from the server to your client. If the catalog is stored in a permanent SAS library, you must define a libref before specifying the PROC DOWNLOAD statement, and you must specify the catalog's two-level name.

To download all of the catalogs in a SAS library, specify INCAT=libref._ALL_.

If you specify this form for the INCAT= option, you must specify the same form for the OUTCAT= option.

You can transfer catalogs with entries that contain graphics output as well as other catalog entries.

CAUTION
Some catalog entry types are not compatible between SAS releases. If you attempt to download a catalog entry from a server to a client that is running a different SAS release, then the client catalog entry that is being downloaded might not be supported at the client. In this case, the catalog entry will not be transferred and the following error message is displayed:

```
WARNING: FILEFMT entries
```

INDEX=YES | NO
specifies whether to re-create an index at the client when you download a SAS data set. You can specify this option when using the DATA= option (if you omit the OUT= option) or when using the INLIB= and OUTLIB= options.

If you download a single data set and omit the OUT= option, or if you download a SAS library, the index is re-created by default.

If you specify the OUT= option and the DATA= option, the index is not re-created.

Default YES

Restriction If the INDEX=YES and the OUT= option are used together in a PROC DOWNLOAD statement, indexes defined on the DATA= data set will not be re-created on the client.

See "INDEX=YES | NO" on page 223

For syntax information about the SAS data set option INDEX=, see "INDEX= Data Set Option" in *SAS Data Set Options: Reference*.

For conceptual information about SAS data set indexing, see "Understanding SAS Indexes" in *SAS Language Reference: Concepts*.

Example "Example 7: Transfer Data By Using Data Set Options and Attributes" on page 277.

INFILE=*server-file-identifier*

specifies the external file that you want to download from the server to the client.

If you use the INFILE= option, you must also use the OUTFILE= option.

server-file-identifier can be one of the following:

fileref

is used if you have defined a fileref on the server that is associated with a single file. You must define the fileref before specifying the PROC DOWNLOAD statement.

fileref(member)

is used if you have defined a fileref on the server that is associated with an aggregate storage location, such as a directory or a partitioned data set.

member

specifies one or more files in that aggregate storage location. You can use the asterisk character (*) as a wildcard in the *member* specification to download multiple files via a single PROC DOWNLOAD statement. The * matches zero or more characters.

You must define the fileref before specifying the PROC DOWNLOAD statement.

Note: The transfer of hidden files is not supported when using the (*) wildcard

The following examples demonstrate the use of the wildcard character. The fileref in the examples is `loc`.

Table 15.1 *Examples: The Wildcard Character in PROC DOWNLOAD*

`infile=loc('*')`	A single asterisk specifies all of the files in the aggregate location.	`all files`
`infile=loc('*dat')`	A leading asterisk specifies all files that end with the same characters. The example selects all files that end with `dat`.	`testfile.dat` `report.old.dat`

`infile=loc('test*')`	A trailing asterisk specifies all files that begin with the same characters.
	The example selects all files that begin with `test`.
		`test.dat`
		`testfile.history`
		`test.tar.gz`

`infile=loc('t*file')`	An embedded asterisk specifies all files that have both the same beginning and ending characters. The example selects all files that begin with `t` and end with `file`.	`tst_1_file` `tst_2_file`
`infile=loc('f*.txt')`	An asterisk can represent the `NULL` string.	`f.txt`

The example below shows how to use a wildcard to transfer all files whose filename starts with the letter **`f`** and which have an extension of **`.sas`**. The specified files will be downloaded from the **`/user/progs`** directory on a UNIX server to the **`c:\Users\test`** directory on a Windows client.

See "FILENAME" on page 201

Example
```
filename locHost 'c:\Users\test';
rsubmit;
    filename remHost '/user/progs';
    proc download infile=remHost('f*.sas')
                  outfile=locHost;
    run;
endrsubmit;
```

Example "Example 2: Use a FILENAME Statement with the UPLOAD and DOWNLOAD Procedures " on page 203.

'external-file-name'
is used to explicitly define the file that is to be downloaded.

```
infile='filename.txt'
```

INLIB=*server-SAS-library*
specifies a SAS library to download from the server to the client. All three forms of this option are equivalent. This option must be used with the OUTLIB= option (in any of its forms). Before using this option, you must define the libref that is used for *server-SAS-library*.

Alias INDD=, IN=

MEMTYPE=(*mtype-list*)
specifies one or more member types to be downloaded.

Here are the valid member types:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias MTYPE=, MT=

Requirement To use this option, you must also specify the INLIB= and OUTLIB= options.

OUT=*client-SAS-data-set* <(*SAS-data-set-options*)>
names the SAS data set on the client that you want the downloaded data set written to. If you want to create a permanent SAS data set, you must define the libref before specifying the PROC DOWNLOAD statement, and you must specify a two-level SAS data set name.

The OUT= option is a valid form of the OUTLIB= option. The DOWNLOAD procedure determines the meaning of the OUT= option as follows:

- If you specify the DATA= option and the OUT= option, the OUT= option names the output SAS data set.

  For example, if the USER= option is set to Mylib, the following statement downloads the data set A from the library Mylib on the server to the library Mylib on the client:

```
proc download data=a out=a;
run;
```

- If you specify only the OUT= option, the DOWNLOAD procedure downloads the last SAS data set that was created on the server.

  For example, the following statement downloads the last data set that was created on the server to the data set Mydata in the library Mylib on the client (assuming USER=Mylib).

```
proc download out=mydata;
run;
```

- If you specify the INLIB= option and the OUT= option, the OUT= option specifies the name of a SAS library.

  For example, the following statement downloads all of the data sets and catalogs that are in the library A on the server to the library RmtLib on the client:

```
proc download inlib=a out=rmtlib;
 run;
```

For details about the effect of omitting the OUT= option, see “Details” on page 267.

See “Specify Data Set Options for the DATA= and OUT= Options in PROC UPLOAD and PROC DOWNLOAD” on page 231

SAS Data Set Options: Reference

“DATA=*server-SAS-data-set* <(*SAS-data-set-options*)>” on page 260

OUTCAT=*client-SAS-catalog*
names the SAS catalog on the client that you want the downloaded catalog written to. If you want to create a permanent SAS catalog, you must define the libref before specifying the PROC DOWNLOAD statement, and you must specify a two-level SAS catalog name. To download all of the catalogs in a SAS library, specify OUTCAT=libref._ALL_.

TIP If you transfer a catalog that contains entries of type PROGRAM, you must compile the entries on the target operating environment before execution. To compile all the PROGRAM entries in a catalog, submit (or remotely submit) the following statements:

```
proc build cat=libref.member-name batch;
    compile;
run;
```

libref identifies the SAS library that contains the catalog and *member-name* identifies the catalog.

Requirement If you specify the OUTCAT= option, you must also specify the INCAT= option. If you specify _ALL_ in the OUTCAT= option, you must also specify _ALL_ in the INCAT= option.

OUTFILE=*client-file-identifier*

identifies an external file on the client that you want a downloaded external file written to.

client-file-identifier can be one of the following:

fileref

is used if you have defined a fileref on the client that is associated with a single file. You must define the fileref before specifying the PROC DOWNLOAD statement.

fileref(member)

is used if you have defined a fileref on the client that is associated with an aggregate storage location such as a directory. *member* specifies which file in that aggregate storage location should be transferred. You must define the fileref before specifying the PROC DOWNLOAD statement. For details about filerefs for your operating environment, see the appropriate operating environment companion documentation.

Note: If a wildcard (*) is used in the INFILE= option, then OUTFILE=*fileref* should point to an aggregate storage location such as a directory.

'external-file-name'

is used to explicitly define the file that is to be downloaded.

Requirement If you use the OUTFILE= option, you must also use the INFILE= option.

OUTLIB=*client-SAS-library*

names the destination SAS library on your client where the downloaded data sets and catalogs from the server are stored. All three forms of this option are equivalent. Before using this option, you must define the libref that is used for *client-SAS-library*.

Note: The OUT= form of this option is the same as the OUT= option that is used to specify a SAS data set. When you use the OUTLIB= option, the DOWNLOAD procedure determines whether the input option was DATA= or INLIB= and processes the downloaded objects appropriately.

The OUTLIB= option must be used with the INLIB= option, but you can use any form of the OUTLIB= option with any form of the INLIB= option. See the description of the INLIB= option for examples that illustrate some valid pairs of these options.

Alias OUTDD=, OUT=

VIEWTODATA
for a library transfer only, causes view descriptor files to be transferred as data sets instead of as view files, which is the default. If you want some views to be transferred as view files and other views to be transferred as data sets, you would have to perform two separate transfers. If you attempt to use this option for a single data set transfer (by using the DATA= option), an error results.

V6TRANSPORT
specifies that data should be translated by using the SAS 6 "File Format Translation Algorithms" on page 462. Specify this option only when you want to use the SAS 6 translation style explicitly, and both the client and the server run SAS 8 or a later release of SAS.

When V6TRANSPORT is specified, the default behavior is to promote a numeric variable whose length is less than 8 bytes. To prevent a promotion of this length, you can use the EXTENDSN=NO option along with the V6TRANSPORT option.

XATTR=YES | NO
specifies whether to allow for the upload or download of extended attributes that are defined on a SAS data set or SAS library. This option is turned on by default in PROC UPLOAD and PROC DOWNLOAD. The XATTR=YES option is invalid when the OUT= option is specified.

If XATTR=YES is specified with the OUT= option, then XATTR=YES is ignored and a WARNING is sent to the SAS log. For example, the following statement will cause a WARNING to be sent to the SAS log and no extended attributes will be transferred:

```
proc download data=inlib.sales out=outlib.sales xattr=y;
run;
```

Extended Attributes are not transferred when the OUT= option is specified with DATA= on PROC DOWNLOAD or PROC UPLOAD. If the XATTR= option is not specified but the DATA= and OUT= options are, then the data set will be transferred, but no extended attributes will be transferred. For example, the following statement will cause the data set Sales to be transferred without its extended attributes:

```
proc download data=inlib.sales out=outlib.sales;
run;
```

If neither the XATTR= nor the OUT= option is specified on PROC UPLOAD or PROC DOWNLOAD, then extended attributes will be transferred. For example, the following PROC DOWNLOAD statement will cause the data set Sales to be transferred along with its extended attributes:

```
proc download data=inlib.sales;
run;
```

For information about PROC DOWNLOAD options and the default behavior of data set options on data sets being transferred, see Table 14.9 on page 232.

Default YES

Restriction If the XATTR=YES and the OUT= option are used together in a PROC DOWNLOAD statement, then extended attributes defined on the variables in the DATA= data set will not be re-created on the client.

Details

Default Naming Conventions for Downloaded Data Sets

If you omit the OUT= option, which specifies the name of the output data set, from the DOWNLOAD statement, SAS follows these rules to determine the name for the data set:

- If the input data set (the data set that is specified in the DATA= option) has a two-level name and the same libref that is defined for the input data set is also defined in the client environment, then the data set is downloaded to the library on the client that is associated with that libref. The data set has the same member name on the client.

  For example, suppose you submit the following statement:

  ```
  libname orders
     client-SAS-library;
  ```

  If you remotely submit the following statements, the data set Orders.Qtr1 is downloaded to Orders.Qtr1 on the client.

  ```
     /*****************************************/
     /* The libref ORDERS is defined on both  */
     /* the client and server.                */
     /*****************************************/
  libname orders
     server-SAS-library;
  proc download data=orders.qtr1;
  run;
  ```

- If the input data set has a two-level name but the libref for the input data set is not also defined in the client environment, then the data set is downloaded to the default library on the client. This is usually the Work library, but the library might also be defined by using the USER libref.

  The data set retains the same data set name that it had on the server. For example, if you remotely submit the following statements, the data set is downloaded to Work.Qtr2 on the client.

  ```
     /*****************************************/
     /* The libref ORDERS is defined only on  */
     /* the server.                           */
     /*****************************************/
  libname orders
     server-SAS-library;
  proc download data=orders.qtr2;
  ```

```
run;
```

- If the input data set has a one-level name and the default libref on the server also exists on the client, the data set is downloaded to that library.

  For example, suppose you submit the following statement:

```
libname orders
   client-SAS-library;
libname local
   client-SAS-library;
   /***********************************/
   /* This option has no effect in     */
   /* this case.                       */
   /***********************************/
options user=local;
```

  If you remotely submit the following statements, the data set Orders.Qtr1 is downloaded to Orders.Qtr1 on the client.

```
   /*****************************************/
   /* The libref ORDERS is defined on both    */
   /* hosts.                                  */
   /*****************************************/
libname orders
   server-SAS-library;
options user=orders;
proc download data=qtr1;
run;
```

- If the input data set has a one-level name and the default libref on the server does not exist on the client, then the data set is downloaded to the default library on the client. That is, the USER libref on the client is used only if the USER libref on the server does not exist on the client.

  For example, suppose you submit these statements:

```
libname local
   client-SAS-library;
options user=local;
```

  When you remotely submit the following statements, the data set Orders.Qtr1 is downloaded to Local.Qtr1 on the client.

```
   /*****************************************/
   /* The libref ORDERS is defined only on    */
   /* the servers.                            */
   /*****************************************/
libname orders
   server-SAS-library;
options user=orders;
proc download data=qtr1;
run;
```

- If you omit the DATA= option, the last data set that was created on the server during the SAS session is downloaded to the client, as follows:

```
proc download;
run;
```

  The naming conventions on the client follow one of the previously described rules, based on how the last data set was created.

WHERE Statement

Selects observations from SAS data sets.

Restriction: The DOWNLOAD procedure processes WHERE statements when you transfer a single SAS data set.

See: *SAS DATA Step Statements: Reference*.

Syntax

WHERE *where-expression-1 <logical-operator where-expression-n>*;

Required Arguments

where-expression-1
is a WHERE expression.

logical-operator
is one of the following logical operators:

- AND
- AND NOT
- OR
- OR NOT

where-expression-n
is a WHERE expression.

To understand when using the SUBSTR function causes an index to be used, look at the format of the SUBSTR function in a WHERE statement:

```
where substr(variable, position, length)
   = 'character-string';
```

An index is used in processing when all of the following conditions are met:

- *position* is equal to 1
- *length* is less than or equal to the length of *variable*
- *length* is equal to the length of *character-string*

The following example illustrates using a WHERE statement with the DOWNLOAD procedure. The downloaded data set contains only the observations that meet the WHERE condition.

```
proc download data=revenue out=new;
   where origin='Atlanta' and revenue < 10000;
run;
```

For details, see *SAS DATA Step Statements: Reference*.

EXCLUDE Statement

Excludes library members or catalog entries from downloading.

Syntax

EXCLUDE *lib-member-list* </ MEMTYPE=*mtype* >;
EXCLUDE *cat-entry-list* </ ENTRYTYPE=*etype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= options in the PROC DOWNLOAD statement. Use the format *cat-entry-list* </ ENTRYTYPE=*etype*> when you specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

lib-member-list
specifies which library members to exclude from downloading. You can name each member explicitly or use one of the following forms:

prefix:
specifies all members whose names begin with the character string *prefix*. For example, if you specify `TEST:`, all members with names that begin with the letters `TEST` are excluded.

first -last
specifies all members whose names have a value between *first* and *last*. For example, if you specify `TEST1-TEST3`, any files that are named `TEST1`, `TEST2`, or `TEST3` are excluded.

Restriction *first* and *last* must begin with identical character strings and must end in a number.

cat-entry-list
specifies which catalog entries to exclude from downloading. Each element of *cat-entry-list* has the form *entry*.*type*.

entry
is the name of an entry in the *catalog* to exclude from downloading.

.type
is the type of the catalog entry. This part of the name is optional.

MEMTYPE=*mtype*
specifies a member type to exclude from downloading.

Here are the valid member types:

- ALL
- CATALOG
- DATA

- MDDB
- VIEW

Alias MTYPE=, MT=

Requirement To use this option, you must also specify the INLIB= and OUTLIB= options in the PROC DOWNLOAD statement.

ENTRYTYPE=*etype*
specifies a catalog entry type to exclude from downloading. Examples of catalog entry types include FORMAT and DATA.

Alias ETYPE=, ET=

Requirement To use this option, you must specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

SELECT Statement

Selects specific library members or catalog entries to download.

Restriction: You cannot use both the EXCLUDE and SELECT statements in the same PROC DOWNLOAD step.

Note: The SELECT statement also enables you to maintain an ordering and grouping of catalog entries that contain graphics output, because entries are downloaded into the client SAS catalog in the order in which you specify them in the SELECT statement.

Syntax

SELECT *lib-member-list* </ MEMTYPE=*mtype*>;
SELECT *cat-entry-list* </ ENTRYTYPE=*etype*>;

Syntax Description

Use the format *lib-member-list* </ MEMTYPE=*mtype*> when you specify the INLIB= and OUTLIB= options in the PROC DOWNLOAD statement. Use the format *cat-entry-list* </ ENTRYTYPE=*etype*> when you specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

lib-member-list
specifies which library members to download. You can name each member explicitly or use one of the following forms:

prefix:
specifies all members whose names begin with the character string *prefix*. For example, if you specify `TEST:`, all members with names that begin with the letters `TEST` are selected for downloading.

first-last
specifies all members whose names have a value between *first* and *last*. For example, if you specify `TEST1-TEST3`, any files that are named `TEST1`, `TEST2`, or `TEST3` are selected for downloading.

Restriction *first* and *last* must begin with identical character strings and must end in a number.

cat-entry-list
specifies which catalog entries to download. Each element of *cat-entry-list* has the form *entry*.*type*.

entry
is the name of an entry in the *catalog* to download.

.type
is the type of the catalog entry. This part of the name is optional.

MEMTYPE=*mtype*
specifies a member type to download.

Here are the valid member types:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias MTYPE=, MT=

Requirement To use this option, you must also specify the INLIB= and OUTLIB= options in the PROC DOWNLOAD statement.

ENTRYTYPE=*etype*
specifies a catalog entry type to download. Examples of catalog entry types include FORMAT and DATA.

Note: The SELECT statement also enables you to maintain an ordering and grouping of catalog entries that contain graphics output, because entries are downloaded into the client SAS catalog in the order in which you specify them in the SELECT statement.

Alias ETYPE=, ET=

Requirement To use this option, you must specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

TRANTAB Statement

Specifies the translation table to use when translating character data for a download from the server to the client.

Restriction: You can specify only one translation table per TRANTAB statement. To specify additional translation tables, use additional TRANTAB statements.

Requirement: To use the TRANTAB statement, you must specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

See: *SAS National Language Support (NLS): Reference Guide*

Syntax

TRANTAB NAME=*translation-table-name*
<*options*>;

Usage: DOWNLOAD Procedure

Using: DOWNLOAD Procedure

VALIDMEMNAME and VALIDVARNAME System Options

If the data that you are transferring contains an invalid SAS name, such as a name containing special characters, national characters, or embedded blanks, then you can specify VALIDVARNAME=ANY or VALIDMEMNAME=EXTEND before the sign-on statement to successfully transfer the files. The following types of data can contain nonstandard SAS names when you use the VALIDVARNAME and VALIDMEMNAME system options with PROCS UPLOAD and DOWNLOAD:

- a SAS data set
- a SAS library
- a SAS variable
- a DBMS table
- a DBMS table column heading

Note: You must specify the VALIDMEMNAME and VALIDVARNAME system options before the SIGNON statement.

For more information about these Base SAS system options, see "VALIDMEMNAME=" in *SAS System Options: Reference* and "VALIDVARNAME=" in *SAS System Options: Reference*.

Results: DOWNLOAD Procedure

Results: DOWNLOAD Procedure

The DOWNLOAD procedure writes a series of informative messages to the SAS log when it executes. Examples of these messages are shown in the following output.

Output 15.1 *SAS Log Messages from the DOWNLOAD Procedure*

```
NOTE: Remote submit to B commencing.
 1    proc download outfile='client-external-file'
 2       infile='server-external-file';run;
 NOTE: TEXT download in progress from
       infile=server-external-file to
       outfile=client-external-file
 NOTE: Downloaded 4 records and 136 bytes.
 NOTE: 4 records were written to the file client-external-file.
       The maximum record length was 65.
       The minimum record length was 0.
 NOTE: 136 bytes were transferred at 136 bytes/second.
 NOTE: The PROCEDURE DOWNLOAD used 0.05 CPU seconds and 1455K.

 NOTE: Remote submit to B complete.
$
```

Examples: DOWNLOAD Procedure

Example 1: DTS: Transfer Data Using WHERE Statements

The UPLOAD and DOWNLOAD procedures process WHERE statements and the WHERE= data set option when you transfer a single SAS data set. Because the transferred data set contains only the observations that meet the WHERE condition, transfer time is minimized.

```
signon foo sascmd="!sascmd -nosyntaxcheck";
```

```
data school;
length name $ 20 class $1;
input name class amount;
cards;
Tom K 30
Sue 1 10
Ab K 3
;

rsubmit status=no;
  proc upload data=school out=kindergarten;
    where class='K';
  run;
endrsubmit;
```

Example 2: DTS: The MEMTYPE= Option in the SELECT Statement

This example downloads the catalogs Names and Salary and the data set Media in the data library Remlib on the server and stores them in the library Loclib on the client:

```
rsubmit;
  proc download inlib=remlib outlib=loclib;
    select names salary media(mt=data) / memtype=cat;
  run;
endrsubmit;
```

Example 3: The ENTRYTYPE= Option in the EXCLUDE Statement in PROC DOWNLOAD

This example downloads all catalog entries that are in the catalog Remote.Main_Formats on the server, except the format entries XYZ and GRADES. It then stores them in the catalog Local.Secondary_Formats on the client:

```
rsubmit;
  proc format lib=remote.main_formats;
    value grades 1='one';
    value aformat 1='one';
    value xyz 1='one';
  run;
  proc download incat=remote.main_formats
    outcat=local.secondary_formats;
    exclude xyz grades / entrytype=format;
  run;
endrsubmit;
```

Example 4: The ENTRYTYPE= Option in Two SELECT Statements in PROC DOWNLOAD

This example maintains the original ordering and grouping when transferring catalog entries that contain graphics output. Assume that you have a catalog named FINANCE that has two entries that contain graphics output, Income and Expense. You want to download the two catalog entries that contain graphics output in the order in which they are stored on the server. That is, you want Income to appear before Expense, not alphabetically as the DOWNLOAD procedure would normally transfer them. In addition, you have some catalog entries that are grouped by the name Group1, and you want to preserve the grouping when the entries are downloaded.

Remotely submit the following program to transfer these entries in the order that you specify in the first SELECT statement and in the group that you specify in the second SELECT statement:

```
rsubmit;
  proc catalog cat=mylib.finance;
    copy out=work.remcat entrytype=grseg;
  run;
  proc catalog cat=work.remcat;
    change G3D= income /entrytype=grseg;
    change GPLOT=expense/et=grseg;
    change TEMPLATE=GROUP1/et=grseg;
  run;
  proc download incat=work.remcat outcat=work.loccat;
    select income expense entrytype=grseg;
    select group1 et=grseg;
  run;
endrsubmit;
proc catalog cat=work.loccat;
   contents;
run;
```

Example 5: Inherit Generation Specific Attributes

During library transfers and single data set transfers when OUT= is not specified, data set attributes are inherited in the output data set. In SAS releases after SAS 6, the maximum number of generations is a new inherited attribute. In addition, the next generation number attribute is inherited ONLY when a library transfer occurs. This attribute is inherited only when the generations are actually transferred, and therefore it is NOT inherited for any single data set transfers. In the following example, both the maximum number of generations and the next generation number attributes are inherited in the output data set, because this is a library transfer.

```
rsubmit;
  proc download in=remote out=local;
    select sales(mt=data);
```

```
  run;
```

In the following example, only the maximum number of generations attribute is inherited. The next generation number attribute is not inherited, because this is a single data set transfer, and therefore no generations are transferred.
In the following example, only the maximum number of generations attribute is inherited. The next generation number attribute is not inherited, because this is a single data set transfer, and therefore no generations are transferred.

```
  proc download data=remote.sales;
   run;
endrsubmit;
```

Example 6: Transfer Long Member Names

SAS/CONNECT supports the transfer of long member names, as long as the operating environment supports long member names. This example uses PROC UPLOAD to transfer a data set and a catalog that have long member names, and uses PROC DOWNLOAD to transfer a data set that has a long member name.

```
rsubmit;
  proc upload in=work out=sasuser;
    select longdatasetname(mt=data)
    cat longcatalogname/mt=cat;
  run;
endrsubmit;

data x.sas_institute_employee_data;
set empdata;
run;

rsubmit;
  proc download inlib=x outlib=work;
  run;
endrsubmit;
```

Example 7: Transfer Data By Using Data Set Options and Attributes

PROC UPLOAD and PROC DOWNLOAD permit you to specify SAS data set options in the DATA= and OUT= options. Note that SAS data set options are not supported when using the INLIB= and OUTLIB= options, even when you upload only data sets. The data set options must be associated with a specific SAS data set, so they must be used in the DATA= or OUT= options. For details about additional restrictions, see UPLOAD Procedure on page 217 and DOWNLOAD Procedure on page 255.

Note: Because the OUT= option is not specified, the transferred data set inherits all the characteristics of the input data set except for the index (because the INDEX=NO option is specified).

This example illustrates using the DATA= option and the INDEX=NO option. It also shows the use of the RENAME= and DROP= SAS data set options.

```
rsubmit;
  data survey(compress=yes index=(comments));
  r='response';
  comments='comments';
  x=1;
  run;

  proc download data=survey
    (rename=(r=response) drop=comments)
    index=no;
  run;
endrsubmit;
```

Example 8: Transfer Data Set Integrity Constraints

Integrity constraints are a set of data validation rules that preserve the consistency and correctness of the stored data. These rules are defined by the applications programmer and are enforced by SAS for each request to modify the data.

PROC UPLOAD and PROC DOWNLOAD permit a transferred SAS data set to inherit the characteristics of the input data set. If the OUT= option is omitted when transferring a specific SAS data set, the transferred data set inherits the characteristics of the input data set. A transferred data set also inherits the characteristics of the input data set if it is part of a library transfer. For details about the INLIB= and OUTLIB= options for PROC UPLOAD, see UPLOAD Procedure on page 217. For details about PROC DOWNLOAD, see DOWNLOAD Procedure on page 255. PROC UPLOAD and PROC DOWNLOAD apply integrity constraints to the transfer of data sets. As with other data set characteristics, integrity constraints are inherited by a transferred data set under specific conditions. The only exception is that, if the input file has an index defined and the user specifies the INDEX=NO option, any integrity constraints that are defined for the input file are not inherited. Also, referential integrity constraint types are not transferred when the referential constraints reside in a different library

This example downloads the SAS data set *Rem* in the library Work on the server to the library Work on the client. Any non-referential integrity constraints that are defined for the input data set are inherited by the output data set.

```
rsubmit;
  proc download data=rem;
  run;
endrsubmit;
```

Example 9: The INDEX=NO Option in the PROC DOWNLOAD Statement

This example downloads the SAS data set *Students* in the library Work on the server to the library Work on the client. Any non-referential integrity constraints that

are defined for the input data set are inherited by the output data set unless there are indexes defined on the input data set. In that case, no integrity constraints are defined for the output data set.

```
rsubmit;
  proc download data=students index=no;
  run;
endrsubmit;
```

Example 10: The EXTENDSN= Option in the PROC DOWNLOAD Statement

This example downloads the catalog SCAT in the directory Remote on the server to the directory Work on the client. By default, catalog transfers promote the length of short numerics within SCREEN entry types. This behavior can be overridden by specifying EXTENDSN=NO on the catalog transfer download. The EXTENDSN= option is supported by catalog transfer of SCREEN entry types only.

Note: The V6TRANSPORT option is unnecessary when transferring a catalog.

```
rsubmit;
  proc download incat=remote.scat outcat=work.scat
    extendsn=no;
  run;
endrsubmit;
```

Example 11: Combining Data from Multiple Server Sessions

Using SAS/CONNECT to connect to multiple servers, you can access data on several servers, combine that data on the client, and analyze the combined data. For example, if you have data that is stored under z/OS in a DB2 database and related data that is stored in an Oracle database under UNIX, you can use SAS/CONNECT in combination with SAS/ACCESS to combine that data on your client. This example uses salary and employee data gathered from two servers to illustrate the process. This example signs on to two servers, downloads data from both servers, and performs analyses of the data on the client. The program uses the SIGNON and RSUBMIT statements.

Note: Bullets 2 through 5 apply to downloading both DB2 and Oracle data.

```
      /************************************/
      /* connect to z/OS                  */
      /************************************/
1  options comamid=tcp;
     filename rlink
```

```
      '!sasext0\connect\saslink\tcptso.scr';
    signon zoshost;
      /*************************************/
      /* download DB2 data views using     */
      /* SAS/ACCESS engine                 */
      /*************************************/
2   rsubmit zoshost;
3     libname db db2;
4     proc download data=db.employee
        out=db2dat;
        run;
5   endrsubmit;

      /*************************************/
      /* connect to UNIX                   */
      /*************************************/
6   options
      remote=hrunix comamid=tcp;
        filename rlink
          '!sasext0\connect\saslink\tcpunix.scr';
        signon;

      /*************************************/
      /* download Oracle data using        */
      /* SAS/ACCESS engine                 */
      /*************************************/
2   rsubmit hrunix;
3     libname oracle user=scott password=tiger;
4     proc download
        data=oracle.employee out=oracdat;
        run;
5   endrsubmit;

      /*************************************/
      /* sign off both links               */
      /*************************************/
7   signoff hrunix;
      signoff zoshost cscript=
      '!sasext0\connect\saslink\tcptso.scr';

      /*************************************/
      /* join data into SAS view           */
      /*************************************/
8   proc sql;
      create view joindat as
      select * from db2dat, oracdat
       where oracdat.emp=db2dat.emp;

      /*************************************/
      /* create summary table              */
      /*************************************/
9   proc tabulate data=joindat
        format=dollar14.2;
        class workdept sex;
        var salary;
        table workdept*(mean sum) all,
```

```
        salary*sex;
        title1 'Worldwide Inc. Salary Analysis
                by Departments';
        title2 'Data Extracted from Corporate
                DB2 Database';
    run;

/* display graphics */
10 proc gchart data=joindat;
      vbar workdept/type=sum
        sumvar=salary
        subgroup=sex
        ascending
        autoref
        width=6
        ctext=cyan;
      pattern1 v=s c=cyan;
      pattern2 v=s c=magenta;
      format salary dollar14.;
      title1 h=5.5pct f=duplex
         c=white
        'Worldwide Inc. Salary Analysis';
      title2 h=4.75pct f=duplex
        c=white
        'Data Extracted from Corporate DB2
         Database';
   run;
   quit;
```

1 To sign on to a server, you need to provide several items of information:

- the *server-ID*, which is specified in a REMOTE= system option or as an option in the SIGNON statement.
- the communications access method, which is specified by using the COMAMID= system option in an OPTIONS statement.
- the script file to use when signing on to the server. This script file is usually associated with the fileref RLINK. Using this fileref is the easiest method for accessing the script file.

After you provide all the necessary information, you can submit the SIGNON statement. You can specify the *server-ID* in the SIGNON statement. If you omit the *server-ID* from the RSUBMIT statement, the statements are submitted to the server session that was identified most recently in a SIGNON statement, in an RSUBMIT statement or command, or in a REMOTE= system option.

2 After you connect to two or more sessions, you can remotely submit statements to any of the servers by simply identifying in the RSUBMIT statement which server should process the statements. After the *server-ID* has been specified by a previous statement or option, you are not required to specify it again in the REMOTE statement. However, this example includes the *server-ID* in the RSUBMIT statements, even though the *server-ID* is not required, to clarify which server is processing each group of statements.

3 Associate a libref with the library that contains the DB2 database on the server.

4 The data from the DB2 database can then be downloaded to the client. Note that when you download a view of a database, a temporary SAS data set is

materialized from the view and downloaded to the client. In this example, the output data set on the client is a temporary SAS data set.

5 The ENDRSUBMIT statement ends the block of statements that are submitted to the server.

6 To establish a second server session, set the REMOTE= and COMAMID= options to values that are appropriate for the second server. You also need to set the fileref RLINK again to associate it with the script file for the second server.

7 Terminate the links to both the UNIX server and the z/OS server. Use the CSCRIPT= option to identify the script file for signing off the z/OS server.

8 On the client, you can now use the SQL procedure to join into a single view the two SAS data sets that were created when you downloaded the views from the server.

9 To analyze the joined data, use the name of the view on the client in a PROC TABULATE step.

10 If you have SAS/GRAPH on your client, you can also use graphics procedures to analyze the view that is created from the two server databases.

Example 12: Compute Services and Data Transfer Services Combined: Process in the Client and Server Sessions

Regardless of the motivation for reducing the amount of data that is transferred, incorporating Compute Services will achieve your goal. Compute Services enables you to format and pre-process data into a subset or a summarized form in the server session before transferring the subsequent smaller amount of data to the client session. This balances the use of CPU cycles between the client and server sessions and minimizes the amount of data contributing to network traffic. If you need information from data that is stored on a remote computer, and you do not want to move a copy of the data to the client, you can benefit from combining Compute Services and Data Transfer Services. Reasons for not moving a copy of the data might include the following: The amount of data is too large, the data is frequently updated, and data duplication is to be avoided.The SAS/CONNECT statements SIGNON, SIGNOFF, RSUBMIT, and ENDRSUBMIT enable you to submit statements from a client session to a server session. You can include these statements in a SAS program and do both client and server processing within a single SAS program. This program can be run in an interactive line mode SAS session, in a non-interactive SAS session, or by including the program in a client session. In each case, the program executes statements in both the client and server sessions.This program processes data on a server, downloads the resulting SAS data set, creates a permanent data set in the client session, and prints a report in the client session. You have several choices for running this program:

- Type and submit each line in an interactive line mode SAS session. All of the statements between the RSUBMIT and ENDRSUBMIT statements are submitted to the server session for processing. All other statements are processed in the client session.

Note: When statements are submitted to the server session, several statements can be grouped into a single packet of data that is sent to the server session. Therefore, a line that is remote submitted is not necessarily processed immediately after you enter it in the client session.

- Build a file that contains all these statements, and use a %INCLUDE statement to include the file in an interactive line mode session. The file is processed immediately.
- Build a file that contains all these statements and run a non-interactive SAS job to process the statements as follows:

```
sas file-containing-program
```

- Build a file that contains all these statements, and use an INCLUDE command to include the file. You must submit the included statements from the windowing environment.
- Build a file and issue the SUBMIT command from the Explorer window. For details, see "Use SAS Explorer to Monitor SAS/CONNECT Tasks" on page 40.

```
      /************************************/
      /* prepare to sign on               */
      /************************************/ 1
 options
      comamid=tcp
      remote=netpc;  2
 libname lhost 'c:\sales\reg1';

      /************************************/
      /* sign on and download data set    */
      /************************************/ 3
 signon; 4
 rsubmit; 5
    libname rhost 'd:\dept12'; 6
      proc sort data=rhost.master
         out=rhost.sales;
         where gross > 5000;
         by lastname dept;
      run;
 7
    proc download data=rhost.sales
         out=lhost.sales;
      run; 8
 endrsubmit;

 9
      /************************************/
      /* print data set in client session */
      /************************************/
 proc print data=lhost.sales;
 run;
```

1 Specifies the COMAMID= and the REMOTE= system options in an OPTIONS statement. These two system options define the connection between the client and server sessions.

2 Defines a libref for the SAS library in the client session to identify the location of the data set to be downloaded.

3 Signs on to the server session. The *server-ID* was specified in the preceding OPTIONS statement.

Note: A script file is not used.

4 Uses the RSUBMIT and ENDRSUBMIT statements to define statements to send to the server for processing. If the client session is connected to multiple active server sessions, specifying the server ID in the RSUBMIT statement clarifies which server session should process the block of statements. If *server-ID* is omitted, RSUBMIT directs the statements to the most recently identified server session.

5 Defines the libref for the SAS library in the server session.

6 Creates the Rhost.Sales data set as a sorted subset of the Rhost.Master data set.

7 Transfers the Sales data from the library in the server session (Rhost) to the library in the client session (LHOST).

8 Marks the end of the block of statements to be submitted to the server session. Statements that follow the ENDRSUBMIT statement are processed in the client session.

9 Reads and prints the SAS data set that was downloaded in the PROC DOWNLOAD step.

Example 13: Compute Services and Data Transfer Services Combined: Sort and Merge Data

When multiple client sessions need to access a single data set on the server, Data Transfers Services can be used to distribute the subset of data that is needed by each session. Each client session receives only the data that it needs, and uses Compute Services to process its data in its session. When you use this method, client sessions do not continually access the data set on the server.

This SCL program fragment distributes a data set that contains reservations data from a server that is located at a central office to clients at several franchise offices. The program enables distribution of selected reservations to a franchise office by using a WHERE statement.

```
submit continue;
signon atlanta;

 rsubmit;
      libname mres "d:\counter";
      libname backup "d:\counter\backup";
1
     proc upload data=mres.reserv
          out=combine status=no;
          where origin="Atlanta";
       run;
2
     proc sort data=combine;
```

```
          by resnum;
        run;
3
     proc copy in=mres out=backup;
          select reserv;
        run;
4
     data mres.reserv;
          update mres.reserv combine;
          by resnum;
        run;
   endrsubmit;

signoff;
```

1 Uploads all reservations for a particular location.

2 Sorts uploaded data sets for merging.

3 Backs up existing data set.

4 Merges new and existing data sets.

16

SAS Component Language (SCL) Functions and Options

Use SCL to Locate and Store Sample Script Files

The system option SASSCRIPT= defines the location of the SAS/CONNECT script files. The value of the SASSCRIPT= system option is a logical name or one or more aggregate storage locations (such as directories or partitioned data sets). Setting the SASSCRIPT= system option automatically generates the SAS system option, SASFRSCR. SASFRSCR is set to the value of a fileref that is used to build a list of scripts for SCL applications. When you establish a link while using SAS/ASSIST, this product uses the information provided by the SASFRSCR option to provide a list of available scripts. You can also build a similar menu of script files for user-written applications by accessing the SASFRSCR system option from an SCL program.

The following SCL program obtains the value of the SASFRSCR system option and uses it to create a list of scripts. For information about the SCL functions that are used in this example, see *SAS Component Language: Reference*.

```
INIT;
return;

MAIN:
      /* Get internally-assigned fileref.      */
```

```
   fileref=optgetc('sasfrscr');

      /* Open the directory (aggregate storage */
      /* location).                            */
   dirid=dopen(fileref);

      /* Get the number of files.              */
   numfiles=dnum(dirid);

      /* Define a custom selection list the    */
      /* length of the number of files and     */
      /* allowing users to make one choice.    */
   call setrow(numfiles,1);
return;

TERM:
      /* Close the directory.                  */
   rc=dclose(dirid);
return;

GETROW:
      /* Display the list of filenames.        */
   filename=dread(dirid,_currow_);
return;

PUTROW:
      /* Get directory pathname.               */
   fullname=pathname(fileref);

      /* Concatenate filename that user selects*/
      /* with directory pathname.              */
   name=fullname ||'/'|| filename;
      /* Other SCL statements to use complete  */
      /* filename stored in name.              */
return;
```

Dictionary

COMAMID SCL Function

Returns a string that contains all of the communications access methods that are valid for the operating environment that the SCL code executes under.

Client: Optional

Server: Optional

Syntax

cval=**COMAMID();**

Syntax Description

cval
a string that contains all of the communications access methods that are valid for the specific operating system.

Details

The COMAMID function returns a string that contains all of the communications access methods that are valid for the operating environment that the SCL code executes under. Each value is separated by a blank. This function is useful for providing a list of communications access methods for users. The list is displayed as determined by the developer. The function merely returns a string of values.

Example

The following program fragment gets the string of communications access methods that are valid for the operating environment that this SCL program executes under. After the string is returned, one way to display the values would be in a list box. Although this example does not include it, you would specify that the list box be filled with the text string `cval`.

```
comlist=makelist();
   str=comamid();
   do i=1 to 10;
      com=scan(str,i,' ');
      if com^=' ' then
         comlist=insertc(comlist,com,i);
   end;
```

RLINK SCL Function

Verifies whether a connection was established between a SAS/CONNECT client and a server session.

Client: Optional

Server: Optional

Syntax

rc=**RLINK**(*'server-ID'*);

Syntax Description

rc
is the return code.

'server-ID'
is the name of the server session (specified by REMOTE= *server-ID*) that is being tested.

Details

The RLINK function verifies whether a connection was established between the SAS/CONNECT client and server sessions.

Example

The following statements use the RLINK function and the server ID REMSESS.

```
rc=rlink('REMSESS');
if (rc=0) then
   _msg_='No link exists.';
else
   _msg_='A link exists.';
```

RSESSION SCL Function

Returns the name, description, and SAS version of a SAS/CONNECT server session.

Client: Optional

Server: Optional

Syntax

cval=**RSESSION**(*n);*

Syntax Description

cval
is the character string that contains the following information:

characters 1 through 17
are the session identifier (REMOTE= *server-ID*).

characters 18 through 57
are the description.

characters 58 through 61
are the number of the server session to get session information for. If no connection exists, the returned value is blank. If a connection exists but no description was specified, characters 58 through 61 in the returned value are blanks.

Details

The RSESSION function returns the session identifier and the corresponding description for a SAS/CONNECT server session. You must have previously defined the description by using the RSTITLE function.

Example

This example loops through four sessions and obtains the server session and description, which is returned by using the RSESSION function. The program puts the descriptions in separate arrays for later use (for example, to display a choice of server sessions to upload to).

```
do i=1 to 4;
   word=rsession(i);
   if word ^=' ' then do;
      remote=substr(word,1,17);
      desc=(substr(word,18,57));
      if rlink(remote) then do;
         if desc=' ' then desc = remote;
         cnt=cnt + 1;
         entrys{cnt}=remote;
         comam{cnt}=desc;
      end;
   end;
end;
```

RSTITLE SCL Function

Defines a description for an existing connection to a SAS/CONNECT server session.

Client: Optional

Server: Optional

Syntax

sysrc=**RSTITLE**(*session-ID, description);*

Syntax Description

sysrc
is 0 if the description was saved or nonzero if the operation failed.

session-ID
is the name of the server session (specified by CONNECTREMOTE= *server-ID*). The string can contain a maximum of eight characters.

description
is a description to associate with the server session. The string can contain a maximum of 40 characters.

Details

The RSTITLE function saves the session identifier and description for an existing connection to a server session. This information can be retrieved by using the RSESSION function to build a list of connections. The list can then be used to select a connection when submitting statements to a server.

Example

The following statements define the description `z/OS Payroll Data` for the remote session by using the identifier `A`:

```
session='A';
descrip='z/OS Payroll Data';
rc=rstitle(session,descrip);
```

17

SAS/CONNECT Script Statements

Dictionary

ABORT Script Statement

Stops execution of a script immediately and signals an error condition.

Syntax

ABORT;

Details

The ABORT statement immediately stops execution of a script and terminates the SIGNON or the SIGNOFF function. ABORT prevents other script statements from executing when the communication link has not been established successfully. When it executes, the ABORT statement signals an error condition, and an error message is issued and displayed in the SAS Log window. To terminate execution of a script under normal conditions, use the STOP statement.

UNIX Specifics: test

CALL Script Statement

Invokes a routine.

Syntax

CALL *label*;

Syntax Description

label
identifies the starting point for executing a block of statements until a Return statement is reached.

Details

The CALL statement causes the statements that are specified after *label* to be executed until a RETURN statement is encountered. When a RETURN statement is reached, script processing resumes at the statement that is specified after the CALL statement.

ECHO Script Statement

Controls the display of characters that are sent from the server while a WAITFOR statement executes.

Syntax

ECHO ON | OFF;

Syntax Description

ON
specifies that the characters are displayed.

OFF
specifies that the characters are not displayed. This is the default.

Details

The ECHO statement is useful when you are debugging a script.

GOTO Script Statement

Redirects execution of a script to the specified script statement.

Syntax

GOTO *label*;

Syntax Description

label
specifies a labeled statement that is located elsewhere in the script.

Details

The GOTO statement can also be written as GO TO.

IF Script Statement

Checks conditions of labeled script statements before they execute.

Syntax

IF *condition* **GOTO** *label*;

IF NOT *condition* **GOTO** *label*;

Syntax Description

condition
is the test that is performed to determine whether a set of statements should be executed.

label
specifies a labeled statement in the script.

Details

The IF statement conditionally jumps to another statement in the script. The IF statement can check two conditions: connection type and whether the script has been called by the SIGNON or the SIGNOFF command.

If the statement is testing for sign-on or sign-off, *condition* should be one of the following:

SIGNON
specifies that the SIGNON command invoked this script.

SIGNOFF
specifies that the SIGNOFF command invoked this script.

If the statement is testing for connection type, *condition* should be either FULL SCREEN or one of the values for the COMAMID= system option.

The value FULLSCREEN can be used to detect any full-screen 3270 connection. The remaining values correspond to values for the COMAMID= system option.

label must specify a labeled statement in the script. For example, in the following IF statement, ENDIT is a label that is followed by one or more statements that terminate the link when the user has issued a SIGNOFF command:

```
if signoff then goto endit;
```

INPUT Script Statement

Displays a prompt to the user that requests a response for the server.

Syntax

INPUT <NODISPLAY> '*prompt*';

Syntax Description

NODISPLAY
is an optional parameter that is used to indicate that the input will not be displayed on the screen. This parameter is commonly used when a user is

prompted to provide a password so that the password is not displayed as it is entered.

'*prompt*'
is a character string and must be enclosed in quotation marks.

Details

The INPUT statement specifies a character string that is displayed to the user when the script executes. The specified string should be a prompt that requests a response from the user, who must respond by pressing Enter or Return (as a minimum response), before script execution can continue. For example, in automatic sign-on scripts, the INPUT statement is used to prompt the user for the user ID and the password that are needed for signing on to the server.

The INPUT statement does not automatically transmit a carriage return or an Enter key. Therefore, when writing a script, if you want to transmit a carriage return or Enter key to the server, you must use a TYPE statement after an INPUT statement.

LOG Script Statement

Sends a message to the client SAS log.

Syntax

LOG '*message*';

Syntax Description

'*message*'
is a text string that must be enclosed in quotation marks.

Details

The LOG statement specifies a message that is written to the SAS log. You can use this statement to issue informative notes or error messages to the user as the script executes. For example, the sample scripts in SAS use the following LOG statement to inform users that the SIGNOFF completed successfully:

```
log 'NOTE: SAS/CONNECT conversation terminated.';
```

NOTIFY Script Statement

Sends a message in a window to the client session.

Syntax

NOTIFY *'message'*;

Syntax Description

'*message*'
is a text string that must be enclosed in quotation marks.

Details

The NOTIFY statement sends a message to the user on the client by creating a window that displays the message. The user must select **Continue** to clear the window. The NOTIFY statement is similar to the LOG statement, but it enables you to highlight messages that might not be noticed in the log.

RETURN Script Statement

Signals the end of a routine.

Syntax

RETURN;

Details

The RETURN statement indicates the end of a group of statements that form a routine in a script. The routine begins with a statement label and is invoked by a CALL statement.

SCANFOR Script Statement

Specifies a pause until conditions are met (an alias for WAITFOR).

Syntax

SCANFOR *pause-specification-1 <pause-specification-2>*…;

Syntax Description

pause-specification
See the description of *pause-specification* in the WAITFOR statement.

Details

The SCANFOR statement is an alias for the WAITFOR statement. See the description of the WAITFOR statement.

STOP Script Statement

Stops execution of a script under normal conditions.

Syntax

STOP;

Details

The STOP statement is used to terminate script execution under normal conditions. Usually, you use the STOP statement at the end of a group of statements that perform sign-on tasks or sign-off tasks.

To halt the execution of scripts under abnormal conditions, use the ABORT statement.

TRACE Script Statement

Controls the display of script statements in the Log window as they execute.

Syntax

TRACE ON | OFF;

Syntax Description

ON
specifies that statements are displayed in the Log window.

OFF
specifies that statements are not displayed in the Log window. This is the default.

Details

The TRACE statement is most useful when debugging a script.

You can set the TRACE statement on or off several times in a script in order to trace execution of selected statements.

TYPE Script Statement

Sends characters to the server as if they were entered at a personal computer.

Syntax

TYPE *text*;

Syntax Description

text
is the user-specified string of characters sent to the server.

Details

Overview of the TYPE Statement

The TYPE statement sends characters to the server as if they had been entered on a personal computer that is attached to that operating environment. For example, in a script that automatically signs on to the server, you use a TYPE statement to issue the server sign-on command.

text can be any combination of the following:

- literal string(s) that are enclosed in quotation marks, such as 'any string'.
- hexadecimal character string(s) that are enclosed in quotation marks, such as '01020304X'.
- 3270 key mnemonics if you have a 3270 connection.

If you use TYPE statements in the script and some characters that are specified by the statement are not entered, then try using the WAITFOR statement to establish a pause in script execution between TYPE statements.

To use a TYPE statement that has more than 80 characters in a sign-on script, divide the TYPE statement into two or more TYPE statements. To divide the TYPE statement, insert a hyphen (-) at the division point. For example, consider the following TYPE statement:

```
type "sas options ('dmr comamid=tcp')"
enter;
```

To divide this statement, change it as follows:

```
type "sas options ('dmr comamid=-" enter;
type "tcp')" enter;
```

Note: Do not insert spaces before or after the hyphen.

ASCII Control Character Mnemonics

To specify an ASCII control character in the TYPE statement, use a mnemonic representation of the character. The following table lists the ASCII control characters and the corresponding mnemonics, decimal codes, and hexadecimal values.

- Do *not* enclose an ASCII mnemonic in quotation marks.
- In the TYPE statement, use only the values from decimal 0 to 127 (hexadecimal 0 to 7F). Do *not* use any of the extended ASCII characters whose values are greater than 127 (decimal).

Table 17.1 *ASCII Character Mnemonics*

ASCII Control Character	Mnemonic Representation	Decimal Value	Hexadecimal Value
Line feed	LF or CTL_J	10	0A

ASCII Control Character	Mnemonic Representation	Decimal Value	Hexadecimal Value
Carriage return	CR or CTL_M	13	0D

WAITFOR Script Statement

Specifies a pause until specific conditions are met.

Syntax

WAITFOR *pause-specification-1<pause-specification-2>*…;

Syntax Description

pause-specification
is the criteria used to determine when the pause is terminated for the WAITFOR statement and processing continues.

The value of *pause-specification* can be either of the following:

time-clause<:timeout-label>

time-clause
specifies a time period in the form *n* SECONDS.

n is the number of seconds that the client waits before processing continues. If you specify 0 SECONDS, a time-out occurs almost immediately. In most cases, you should specify a value greater than 0. You can specify only one time clause in a WAITFOR statement.

:timeout-label
specifies the label of a statement that exists later in the script. The label must be preceded by a colon (:). When you specify a label, script execution passes to the labeled statement after a time-out occurs. If no label is specified, execution proceeds with the statement that is specified after the WAITFOR statement.

text-clause<:text-label>
text-clause
specifies a string that the client waits to receive from the server. The string can be the following

- a character string that is enclosed in quotation marks
- a hexadecimal string that is enclosed in quotation marks

When *text-clause* is specified, SAS on the client reads input from the server, searching for the specified string. With 3270 connections, SAS on the client scans the server screen (instead of reading characters sequentially).

:text-label
specifies the label of a statement that exists later in the script. The label must be preceded by a colon (:). When you specify a label, script execution passes to the labeled statement after a time-out (if the label follows a time clause) or after the specified string has been read (if the label follows a text clause). If no label is specified, execution proceeds with the statement that is specified after the WAITFOR statement.

Details

The WAITFOR statement directs SAS on the client to do one of the following:

- pause for a specified time
- pause for a specified time or until specified characters from the server are received
- pause until specified characters from the server are received

Usually, a WAITFOR statement is used after a TYPE statement sends input to the server that causes the client to wait for the server's response to the input. For example, in the sample scripts, a WAITFOR statement follows the TYPE statement that invokes SAS on the server.

You can include one or more pause specifications in a WAITFOR statement. When you include more than one pause specification, use commas to separate the clauses.

Comparisons

- You must specify either a time clause or a text clause in the WAITFOR statement. Or you can specify multiple text clauses or combine a time clause and one or more text clauses. Labels and screen location specifications are optional.
- If the only specification in the WAITFOR statement is a time clause, there is a pause during the script's execution. When the specified time has elapsed, control passes to the next statement in the script. For example, the following WAITFOR statement causes a 2-second pause in script execution:

  ```
  waitfor 2 seconds;
  ```

- If the WAITFOR statement contains a time clause followed by a label, a pause occurs and control passes to the labeled statement. The following WAITFOR statement causes a 2-second pause and then passes control to the script statement labeled STARTUP:

  ```
  waitfor 2 seconds :startup;
  ```

- If the WAITFOR statement contains a time clause and a text clause, the client waits the specified time for the specified characters from the server. If the client does not receive the expected characters before the time expires, then a time-

out occurs and control passes to the next statement or to the labeled statement (if a label is specified by the time clause). For example, when the following WAITFOR statement executes, the client pauses for 5 seconds and reads any input sent by the server:

```
waitfor 'Enter your password',
   5 seconds :nohost;
```

If the following string is sent by the server within 5 seconds, no time-out occurs and control passes to the next statement in the script:

```
Enter your password
```

If the string is not received within 5 seconds, a time-out occurs and control passes to the statement labeled NOHOST.

- You can specify labels for both text clauses and time clauses. For example:

```
waitfor 'Enter your password' :startlnk,
   5 seconds :nohost;
```

This WAITFOR statement is the same as the preceding example except that a label is specified after the text clause. Therefore, if the following string is sent by the server within 5 seconds, no time-out occurs and control passes to the statement labeled STARTLNK:

```
Enter your password
```

If the string is not received within 5 seconds, a time-out occurs and control passes to the statement labeled NOHOST, as in the previous example.

- If you do not specify a time clause (that is, if you specify only a text clause), a time-out cannot occur, and the client waits indefinitely for the specified text response from the server. Usually, you should specify a time clause to avoid being trapped in an infinite wait.
- If you specify multiple text clauses in a WAITFOR statement, the commas that separate the clauses imply a logical OR operator, so only one of the text clauses needs to be satisfied (true).

PART 4

Administration

18

Access Methods

Access Methods Supported by SAS/CONNECT

Overview

A communications access method is the interface between SAS and the network protocol that you use to connect two operating environments. You must use a communications access method with SAS/CONNECT. The communications access method that you choose is determined by the network protocols that you have available at your site and the operating environments that you are connecting.

SAS/CONNECT uses the *TCP/IP* access method by default for the UNIX and Windows operating environments to establish client/server network connections. The z/OS operating environment can also use TCP/IP. However, by default, the z/OS operating system uses the XMS access method. The XMS access method allows communication only between clients and servers that run within a single z/OS operating environment.

Use the COMAMID= option if you want to use TCP/IP for z/OS or if you choose to explicitly specify TCP/IP for UNIX or Windows. For more information, see “COMAMID=” on page 101.

The following are descriptions of these access methods:

TCP/IP (Transmission Control Protocol/Internet Protocol)
is a program-to-program interface that is supported on hardware from multiple vendors. TCP/IP is supported under the UNIX, z/OS, and Windows operating environments.

XMS (Cross-Memory Services)
is an interface that is part of the z/OS operating environment and is used by programs that run within a single z/OS environment.

For more information, see “MP Connections on z/OS” on page 374.

Note: Before using TCP/IP on z/OS, you must configure your system to run SAS under TCP/IP. Complete the steps outlined in the sections, “System Configuration for Using SAS with TCP/IP” and “Post-Installation Configuration for SAS/CONNECT Software” in the Configuration Guide for SAS 9.4 for Foundation on z/OS.

For more information using TCP/IP, see “TCP/IP Access Method” on page 308.

For information about using TCP/IP in each of the supported SAS/CONNECT operating environments, see the following:

- UNIX on page 341
- z/OS on page 363
- Windows on page 384

TCP/IP Access Method

Overview

TCP/IP is a set of layered protocols that enable processes on the same computer to communicate or processes on different computers to communicate across a network.

Although you might refer to a computer by using its host name, TCP/IP applications refer to computers by using their IP addresses. To facilitate the use of host names in a network, the Domain Name System translates host names to IP addresses. This Domain Name System provides host-to-IP address mapping through network server hosts, which are called domain name servers. The Domain Name System also provides other information about server hosts and networks, such as the TCP/IP services that are available to the server host and the location of the domain name servers in the network.

About TCP/IP Addressing

TCP/IP applications refer to networked computers via their fully qualified domain names (FQDN) and their IP addresses. Because IP addresses can change easily,

SAS applications that contain hardcoded IP addresses are prone to maintenance problems. To avoid such problems, use an FQDN instead of an IP address. The name-resolution system that is part of the TCP/IP protocol is responsible for locating the IP address that is associated with the FQDN.

SAS 9.2 introduced support for the Internet Protocol, IPv6, which is the successor to Internet Protocol, IPv4. Rather than replacing IPv4 with IPv6, SAS now supports both protocols. There will be a lengthy transition period during which the two protocols will coexist. A primary reason for the new protocol is that the limited supply of 32-bit IPv4 address spaces was being depleted. IPv6 uses a 128-bit address scheme, which provides more IP addresses than does IPv4.

Here are examples of an FQDN, an IPv6 address, and an IPv4 address:

FQDN address example: `d6292.us.company.com`

IPv6 address example: `db8::01`

IPv4 address example: `10.23.2.3`

Configure the TCP/IP Services File

Overview

The SERVICES file defines port resources that are used when TCP/IP is used to connect client/server sessions.

A service for each SAS/CONNECT server session must be defined in the SERVICES file on the server and on each client computer that connects to it.

The -SERVICE Option

The -SERVICE option is a spawner start-up option that can be used to specify a defined numeric port value or service name for the spawner being started.

If the spawner is started with the -SERVICE=<*service-name*><*port-number*> syntax, then the *service-name* value or the *port-number* value that was used to start the spawner on the server must be used by the client to sign on.

For information about the -SERVICE option, see -SERVICE.

Note: The spawner service name and port number can be configured in the client's SERVICES file, but this is not a requirement. The port information (service or port) can be defined in metadata or clients can explicitly specify the port information about the command line or in the SIGNON statement.

Services That Require an Entry in the Services File

Here are some examples of port services that require configuration in the **services** file:

- Telnet service
- spawner ports
- firewall computer port
- dedicated TCP/IP port service that is used for MP CONNECT piping

Note: If you have access to a UNIX operating environment, see the `services` manual page for more information about this file.

Location of the Services File

The location of the **services** file depends on the operating environment. Typical locations for the TCP/IP services file are the following:

Operating System	Location
UNIX and z/OS	**/etc/services** See the “services” manual page for more information about this file and its location.
Windows	**C:\Windows\System32\drivers\etc\services**

Rules for Updating TCP/IP Port Numbers and Service Names

To configure your TCP/IP services file to use with the SAS/CONNECT spawner, add an entry to the services file for each SAS server (either local or remote) that you have configured.

Here are the rules for adding the entry:

- The port number that you use should be an unused port number greater than 1024. Any port number equal to or less than 1024 is reserved.
- The protocol must always be TCP.

- The server name can be up to eight characters.
- The first character must be a letter or an underscore.
- Subsequent characters can be letters, numeric digits, underscores, the dollar ($) sign, or the at (@) sign.

Here is the syntax for a typical **services** file.

```
<official-service-name> <port-number/protocol-name >
<alias-name> <service-description>
```

Here a sample services file:

Figure 18.1 *Sample Services File*

```
# Port Services
 telnet             23/tcp                 # Telnet service
 spawnport          4016/tcp               # UNIX SAS/CONNECT spawner port
 mktserve           4017/tcp               # Server for Marketing & Sales
 server1            5011/tcp               # SAS/SHARE server 1
 firewall           5010/tcp               # Firewall machine port
 pipe1              5020/tcp               # MP Connect pipe
 sea                5021/tcp    biscuit    # SAS/SHARE server 3
 # A blank line goes here.
```

Note: You must enter a blank line (press the ENTER key) at the end of the **services** file. If a blank line is not at the end of the file, the final line in the file is not detected. For example, if you run a SAS script that contains the name of the configured SAS/SHARE service `sea`, this error message is displayed:

```
Cannot find TCP service 'sea'
```

Here is an explanation of each field in the example above:

service-name
specifies the name of the service. Service names must meet the criteria for a valid SAS name. (For details about SAS naming rules, see "Rules for Words and Names in the SAS Language" in *SAS Language Reference: Concepts*.) For example, you can create a service named SPAWNER for the UNIX spawner program. You need the Telnet service when signing on to any server that does not use a PC or a UNIX spawner program.

You use the service name as the value for the REMOTE= option or in the SIGNON statement to perform a server sign-on.

port-number
is a unique number that is associated with the service name. Each reference to that service in other node **services** files must match the service's port number exactly. Port numbers 0 through 1023 are reserved for system use. Port numbers that are greater than 1023 are available for user-created services.

protocol-name
identifies the protocol. `udp` and `tcp` are examples of protocol names.

alias-name
is an optional synonym for the service. Alias names can be application- or user-dependent. For example, one application can refer to the server as `sea` and another application can refer to the same server as `biscuit`.

Note: Each client and server must configure the alias in its **services** file before the alias can be used successfully. For example, the service name, `sea` and the

alias `biscuit` must be configured in the **`services`** file of each client and server that uses the alias.

service-description
describes the service.

Configure SAS/CONNECT for Use with a Firewall

Firewall Concepts

firewall
is a controlled gateway between two networks. A firewall limits external client connections to a set of restricted ports on one or more computers that are inside the firewall.

Web servers and other network applications can also use firewalls to limit access to servers. SAS/CONNECT permits TCP/IP connections between clients outside a firewall to a spawner that runs on a SAS/CONNECT server inside a firewall.

socket inheritance
enables the server session to inherit the socket that the spawner uses to communicate with the client session. The socket is then used for subsequent communications between the client and the server session. Socket inheritance is significant because a single port can be used for starting an unlimited number of server sessions.

Before this innovation, a separate port was opened for each client that connected to a server by using a spawner. Socket inheritance limits the number of ports that are used for connections through a firewall, which improves the security of a firewall configuration and simplifies administration of a firewall configuration.

Requirements for Using a Firewall

- The external clients and the servers within the firewall must be running SAS 6.12 TS065 or later.
- The TCP/IP communications access method must be used for establishing a network connection between clients and servers.
- Firewall software must be installed on the server that maintains the separation between the internal network and the Internet.

- A port must be defined on the firewall server to be used as a gateway between external clients and the internal network. The firewall software must route the firewall server port to the predefined server port.
- A spawner must be running on a server inside the firewall. For complete details about the spawner program, see “Introduction to the SAS/CONNECT Spawner” on page 319.

Firewall Configurations

Overview of Firewall Configurations

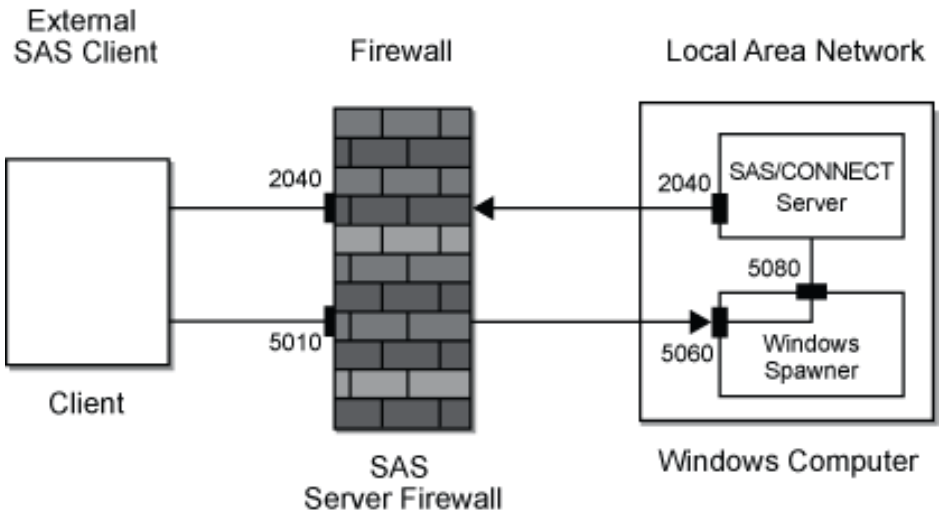
The supported firewall configurations are distinguished by these characteristics:

- A range of restricted ports is available for client/server connections across a firewall.
- A single port is available for all client/server connections across a firewall.

Set Up a Firewall Configuration That Uses Restricted Ports

The example configuration includes an external SAS client, a firewall, and a SAS/CONNECT server session and a spawner program that run on the local area network. Each external client connects to the server using a range of restricted ports.

Figure 18.2 *Firewall Configuration That Uses Restricted Ports*



Here are the steps for setting up a firewall configuration:

1 At each external SAS client, the user must configure the firewall port, 5010, in its services file.

```
fireport                5010/tcp             # Firewall computer port
```

FIREPORT is a defined service in the client's services file that is associated with port 5010. FIREPORT is the single port through which all external SAS clients will access SAS servers in the internal network.

2 The administrator of the firewall server must configure these ports:

- the restricted ports that are used by the external SAS clients and a mapping to the equivalent port numbers on the SAS/CONNECT server
- the firewall port, 5010, and a mapping to 5010 on the SAS/CONNECT server or another port number on the SAS/CONNECT server

Note: Restricted ports are implemented using the TCPPORTFIRST= and TCPPORTLAST= system options that are specified in the SAS start-up file. (See step 4.)

For example, if the external SAS clients use restricted ports 2040 through 2044, the administrator of the firewall server must configure those ports on the firewall server. Also, the administrator must map those ports to the same port numbers on the SAS/CONNECT server.

Specific details about configuring and mapping ports on the firewall server vary according to the specific firewall software that is used.

3 The administrator of the SAS/CONNECT server must configure these ports in its services file:

- the port that is used by the external SAS client to communicate with the spawner
- the ports that are used by the spawner to communicate with the SAS/CONNECT server

Here is an example of these entries in the services file:

```
spawnport          5060/tcp       # Port for external SAS client to spawner
servport           5080/tcp       # Port for spawner and SAS/CONNECT server
```

SPAWNPORT is a defined service in the services file that is associated with port 5060. SERVPORT is associated with port 5080.

4 The administrator of the SAS/CONNECT server must configure one or more restricted ports in the SAS start-up file that executes when the spawner starts the SAS/CONNECT session.

```
sas.exe -tcpportfirst 2040 -tcpportlast 2040 %*
```

In this example, SAS is started and the restricted port is 2040. All communications between external SAS clients and the SAS/CONNECT server are restricted to port 2040.

A range of ports could be specified by increasing the values assigned to the TCPPORTFIRST= and TCPPORTLAST= system options.

5 The administrator of the SAS/CONNECT server must start the spawner using a command that disables socket inheritance:

```
cntspawn -noinheritance -service spawnport -sasdaemonservice servport
   -sascmd mysas.cmd
```

The restricted port that is used by the SAS client and the SAS/CONNECT server is specified in the `mysas.cmd` script via the TCPPORTFIRST= and TCPPORTLAST= system options.

Here is an explanation of the spawner command example above:

Table 18.1 *Explanation of Spawner Command*

Command	Description
`cntspawn`	Starts the spawner
`-noinheritance`	Specifies that sockets cannot be inherited
`-service spawnport`	Specifies that the spawner service will be named 'spawnport,' and that the spawner will listen for requests from SAS clients at port 5060 for connections to a SAS/CONNECT server.
`-sasdaemonservice servport`	Specifies the service or port, 5080, through which the spawner relays the SAS client's request to connect to the SAS/CONNECT server.
`-sascmd mysas.cmd`	Specifies the script that starts the SAS/CONNECT session. The script might contain SAS options that restrict ports.

For details about spawner options, see "Spawner Options" on page 326.

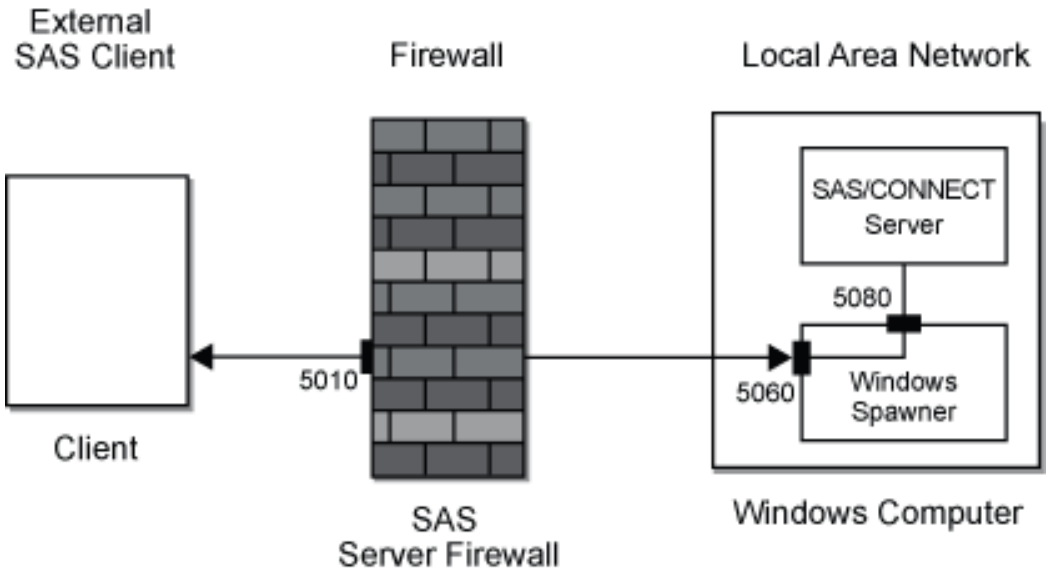
6 To test the configuration, start a SAS session on a computer that is outside the firewall and sign on to the server that is inside the firewall. Here is an example:

```
options comamid=tcp;
signon firewall.fireport username="myuser" password="mypass";
```

Set Up a Firewall Configuration That Uses a Single Port

The example configuration includes an external SAS client, a firewall, and a SAS/CONNECT SAS/CONNECTxternal client connects to the server using a single port, which is enabled by socket inheritance.

Figure 18.3 *Firewall Configuration That Uses a Single Port*



Here are the steps for setting up a firewall configuration:

1 At each external SAS client, the user must configure the firewall port, 5010, in its services file.

```
fireport                5010/tcp             # Firewall computer port
```

FIREPORT is a defined service in the TCP/IP services file that is associated with port 5010. FIREPORT is the single port through which all external SAS clients will access SAS servers in the internal network.

Note: The firewall server does not necessarily have to run SAS software.

2 The administrator of the firewall server must configure the firewall port, 5010, and map it to another port number on the SAS/CONNECT server.

Specific details about configuring and mapping ports on the firewall server vary according to the specific firewall software that is used.

3 The administrator of the SAS/CONNECT server must configure these ports in its services file:

- the port that is used by the external SAS client to communicate with the spawner
- the ports that are used by the spawner to communicate with the SAS/CONNECT server

Here is an example of these entries in the services file:

```
spawnport          5060/tcp       # Port for external SAS client to spawner
servport           5080/tcp       # Port for spawner and SAS/CONNECT server
```

SPAWNPORT is a defined service in the services file that is associated with port 5060. SERVPORT is associated with port 5080.

4 The administrator of the SAS/CONNECT server starts the spawner:

```
cntspawn -service spawnport -sasdaemonservice servport
-sascmd mysas.cmd
```

Note: The command to start the SAS/CONNECT spawner is `CNTSPAWN`.

Here is an explanation of the spawner command:

Table 18.2 *Explanation of Spawner Command*

Command	Description
`cntspawn`	Starts the spawner.
`-service spawnport`	Specifies the service or its port, 5060, at which the spawner listens for requests from SAS clients to connect to a SAS/CONNECT server.
`-sasdaemonservice servport`	Specifies the service or port, 5080, through which the spawner relays the SAS client's request to connect to the SAS/CONNECT server.
`-sascmd mysas.cmd`	Specifies the script that starts the SAS/CONNECT session.

For details about spawner options, see "Spawner Options" on page 326.

5 To test the configuration, start a SAS session on a computer that is outside the firewall and sign on to the server that is inside the firewall. Here is an example:

```
options comamid=tcp;
signon firewall.fireport username="myuser" password="mypass";
```

19

The SAS/CONNECT Spawner

Introduction to the SAS/CONNECT Spawner

Definition

A SAS spawner is a program that starts a SAS session on the server on behalf of a connecting client. The SAS/CONNECT spawner runs on the SAS/CONNECT server, listens for requests, and opens a connection to the server on behalf of the

client. Signing on to the SAS/CONNECT spawner is an alternative to signing on to a server by using a Telnet daemon.

The SAS/CONNECT spawner can listen for requests on multiple ports that are defined in metadata and on a single port that is defined on the spawner invocation command. Starting with SAS 9.4, you can associate multiple SAS/CONNECT servers with a single spawner that is listening on multiple ports.

Spawner invocation options enable you to start and manage the SAS/CONNECT spawner. For more information about these options, see “Spawner Options” on page 326.

For more information about defining multiple ports in metadata using the SAS Deployment Manager, see Add a New Logical Server in an Existing SAS Application Server in the *SAS Intelligence Platform Server Administration Guide*.

For information about the SAS/CONNECT spawner for SAS Viya, see SAS/CONNECT Server and Spawner.

Operating Environment Support for Spawners

SAS 9.4 supports TCP/IP spawners under the UNIX, Windows, and z/OS operating environments. Information about setting up and using the SAS/CONNECT spawner for each of these operating environments can be found in the following locations in this document:

- Chapter 20, “UNIX Operating Environment,” on page 339
- Chapter 21, “z/OS Operating Environment,” on page 361
- Chapter 22, “Windows Operating Environment,” on page 383

For a list of all available spawner invocation options, see “Spawner Options” on page 326.

Benefits of Using a Spawner to Sign On to a Server

Protects Client's User ID and Password

By default, the spawner encrypts the client's user ID and password during sign-on. Without encryption, the user ID and password would pass through the network as clear, readable text, which presents a security risk.

To encrypt all data that flows through the network after sign-on (such as data being processed by remote submits and data transfers), you must use a security service. For details about security services that are supported in SAS 9.4, see *Encryption in SAS*.

Controls Client Access to the Server in a Firewall Configuration

A spawner can be used to control the number of ports that clients outside a firewall can use to access a server that is inside the firewall. Controlled client access facilitates a computer's security and economizes resources. For details, see "Configure SAS/CONNECT for Use with a Firewall" on page 312.

Eliminates the Need for a Sign-On Script

The primary purpose of a sign-on script is to do the following:

- send the user ID and password to the server
- supply the SAS command for starting the SAS session on the server

Because the user ID and password can be directly specified as options in the SIGNON statement (or command), and the spawner controls the start-up of a SAS session on the server, the need for a sign-on script is eliminated.

Use SAS Management Console to Manage the SAS/CONNECT Spawner

Overview

SAS Management Console is the primary administrative user interface for administering SAS servers in the SAS Intelligence Platform. SAS Management Console includes a variety of plug-ins that are used to create and maintain various resources available in the SAS Intelligence Platform. The Server Manager plug-in is used to manage the SAS/CONNECT spawner and SAS/CONNECT server.

Complete documentation for SAS Management Console and the Server Manager plug-in can be found in the following SAS Intelligence Platform documents:

- *SAS Intelligence Platform: Overview*
- *SAS Intelligence Platform: System Administration Guide*
- *SAS Management Console: Guide to Users and Permissions*
- *SAS Intelligence Platform: Application Server Administration Guide*

These documents can be found on the SAS Intelligence Platform Product Documentation page at http://support.sas.com/documentation/onlinedoc/intellplatform/.

Use PROC IOMOPERATE to Manage the SAS/CONNECT Spawner

Overview

You can use the IOMOPERATE procedure to manage the SAS/CONNECT spawner and server. See the IOMOPERATE Procedure in *SAS Intelligence Platform: Application Server Administration Guide* for syntax and detailed information about PROC IOMOPERATE.

PROC IOMOPERATE commands that are used to administer the SAS/CONNECT spawner require a connection to the server. To establish a connection, use the CONNECT command in PROC IOMOPERATE and specify the spawner management port in the URI= option. Once connected, all subsequent PROC IOMOPERATE commands apply to that server until a DISCONNECT or STOP SERVER command is executed. The examples below demonstrate how you can perform various administrative tasks on the SAS/CONNECT spawner and server using PROC IOMOPERATE.

For more information about the SAS/CONNECT spawner -MGMTPORT= option, see "-MGMTPORT <*port-number*>|<'*service-name*'>" on page 328.

> **TIP** After you connect to a server, you can invoke LIST COMMANDS to determine which commands can be run on the server.

Example 1: List Valid PROC IOMOPERATE Commands

The following example demonstrates how you might use the IOMOPERATE procedure, LIST statement, and COMMANDS option to obtain a list of valid commands that can be used in the IOMOPERATE procedure on the specified server. The URI= option in the CONNECT statement specifies the server that you want to connect to and takes the form `iom://hostname:port`.

```
%let cmd=list commands;
proc iomoperate;
  connect uri="iom://hostA:7543;Bridge;USER=jdoe,PASS=abc123";
  &cmd;
quit;
```

Example 2: Display a List of Spawned Servers

The following example demonstrates how you might use the IOMOPERATE procedure, LIST statement, and SPAWNED SERVERS option to show what servers are currently active through the spawner and to show information about clients that are signed on.

```
%let cmd=list spawned servers;
%let spnode=<host-name>;
%let mgmtport=<mgmtport>;
%let mgmtpwd=<password>
%let username=<user-ID>;
/* NOTE: some commands, such as STOP SERVER, require the user-id
*/
/* that is associated with the  spawner process, service, or started
task */

proc iomoperate;
  connect uri="iom://
&spnode:&mgmtport;Bridge;USER=&username,PASS=&mgmtpwd";
  &cmd;
quit;
```

SAS returns output similar to the following:

```
NOTE: The CONNECT command completed.
    &cmd;
Non-server Scriptless SIGNON
    Server class  : 028E4060-D545-11D5-880D-AA0004006D06
    Process owner : sascnn1:61303 (comp.na.abc.com)
    Server id     : 1F97A800-B875-11E4-98DC-F0F6F5C5C1F6
NOTE: The LIST SPAWNED SERVERS command completed.
27   quit;
```

Example 3: Display Spawner Information

The following example demonstrates how you might use the IOMOPERATE procedure, LIST statement, and INFORMATION option to display information about the SAS/CONNECT spawner.

```
%let cmd=list information;
%let spnode=<host-name>;
%let mgmtport=<mgmtport>;
%let mgmtpwd=<password>
%let username=<user-id>;
/* NOTE: some commands, such as STOP SERVER, require the user-id
*/
```

```
/* that is associated with the  spawner process, service, or started
task */

proc iomoperate;
  connect uri="iom://
&spnode:&mgmtport;Bridge;USER=&username,PASS=&mgmtpwd";
  &cmd;
quit;
```

SAS returns output similar to the following:

```
NOTE: The CONNECT command completed.
   &cmd;
36 Information
    CNTSPAWN.Encryption.Algorithms :
    CNTSPAWN.Encryption.FIPS : FALSE
    CNTSPAWN.Encryption.Parameters :
    CNTSPAWN.SpawnerMetadataName :
    CNTSPAWN.UseSecurity : TRUE
    CNTSPAWN.Version.BuildDate : Feb 11 2015
    CNTSPAWN.Version.BuildTime : 20:24:29
    CNTSPAWN.Version.Major : 9
    CNTSPAWN.Version.Minor : 40
    IOM.ClassBase : F917284A-D088-4B97-91F6-BF80B6E7B24D
    IOM.ClassLatest : F917284A-D088-4B97-91F6-BF80B6E7B24D
    IOM.LastCounterReset : 19FEB2015:20:20:19
    IOM.ServerPort : 9700
    IOM.ServerState : Running
    IOM.UniqueIdentifier : B939DC04-B874-11E4-98DC-F0F6F5C5C1F6
    IOM.UpTime : 19FEB2015:20:20:19
    Server.CPUCount : 6
    Server.Command : SAS
    Server.DNSName : COMPA
    Server.FullyQualifiedDNSName : compA.abc.sas.com
    Server.HostKnownBy :  comp.abc.sas.com
    Server.ProcessIdentifier : 16844823
    Server.ProcessOwner : JDOE
    Server.Version : 9.4
    Server.VersionLong : 9.04.01M3P02112015
NOTE: The LIST INFORMATION command completed.
37   quit;
```

Example 4: Display Spawner Attributes

The following example demonstrates how you might use the IOMOPERATE procedure, LIST statement, and ATTRIBUTES option to determine the number of connections made since the SAS/CONNECT spawner started, to determine the number still active, and to determine the number of sign-on failures.

```
%let cmd=list attributes;
%let spnode=<host-name>;
%let mgmtport=<mgmtport>;
%let mgmtpwd=<password>
%let username=<user-id>;
/* NOTE: some commands, such as STOP SERVER, require the user-id
*/
/* that is associated with the  spawner process, service, or started
task */
```

```
proc iomoperate;
  connect uri="iom://
&spnode:&mgmtport;Bridge;USER=&username,PASS=&mgmtpwd";
  &cmd;
quit;
```

SAS returns output similar to the following:

```
NOTE: The CONNECT command completed.
  &cmd;
68 Counters
      CNTSPAWN.ActiveConnectClients : 3
      CNTSPAWN.ActiveConnectMgmtClients : 0
      CNTSPAWN.ActiveConnectServers : 3
      CNTSPAWN.TotalAuthenticationFailures : 0
      CNTSPAWN.TotalConnectClients : 7
      CNTSPAWN.TotalConnectConnections : 7
      CNTSPAWN.TotalConnectServers : 7
      IOM.CounterResets : 0
      IOM.CurrentClients : 1
      IOM.CurrentMemoryUsage : 15716352
      IOM.CurrentThreadCount : 11
      IOM.HighestMemoryUsage : 16056320
      IOM.HighestThreadCount : 12
      IOM.IdleTime : 522.419
      IOM.TimeInCalls : 0
      IOM.TotalCalls : 0
      IOM.TotalClients : 8
NOTE: The LIST ATTRIBUTES command completed.
69   quit;
```

Example 5: Stop and Re-Start the Spawner

The following example demonstrates how you might use the PROC IOMOPERATE procedure to stop and re-start the SAS/CONNECT spawner.

```
%let cmd=stop spawned server id=<server-id>;
%let spnode=<host-name>;
%let mgmtport=<mgmtport>;
%let mgmtpwd=<password>
%let username=<user-id>;
/* NOTE: some commands, such as STOP SERVER, require the user-id*/
/* that is associated with the  spawner process, service, or started
task */

proc iomoperate;
  connect uri="iom://
&spnode:&mgmtport;Bridge;USER=&username,PASS=&mgmtpwd";
  &cmd;
quit;
```

Spawner Options

Introduction

Spawner invocation options consist of SAS/CONNECT spawner options and SAS system options that you can use to run and configure the spawner from the command line. You can use these commands when you invoke the spawner using the CNTSPAWN command in Windows or UNIX, or in a z/OS PARMS file. SAS/CONNECT spawner options fall into 3 categories:

- "General Spawner Options"
- "Security Options"
- "Windows-only Service Options"

If you have a planned deployment of SAS or you used the SAS Deployment Wizard to install any of the SAS Intelligence Platform software, you can also manage the SAS/CONNECT spawner using the PROC IOMOPERATE procedure, beginning with SAS 9.4.

For more information about this procedure, see the IOMOPERATE Procedure in the *SAS Intelligence Platform: Application Server Administration Guide*.

General Spawner Options

Use the following general options with the CNTSPAWN spawner start-up command:

-BINDADDR <*IP_address*>
Specifies an IP address override for the CONNECT Spawner to bind all its listening ports to.

The CONNECT spawner passes the IP address bind override to the CONNECT server that it starts.

If the CONNECT spawner uses its BINDADDR value, then it passes it to the CONNECT server's environment by setting the TCPBINDADDR environment variable for the launched session. The launched CONNECT server then uses this IP address to bind all of its listening ports to.

It is used in multi-NIC (Network Interface Card) environment.

-CLEARTEXT

> **IMPORTANT** With the June 2023 hot fix, the -CLEARTEXT option has been deprecated and is no longer available. For more information, see SAS Note 70114.

allows sign-ons from clients that do not support user ID and password encryption. This option allows clients that are running older releases (prior to SAS 6.09E and SAS 6.11 TS040, which do not support user ID and password encryption) to sign on to the spawner program. Use this option only when absolutely necessary because credentials are transmitted unencrypted. The default encodes all communications.

-DEBUG
turns on debug level output.

-HELP
specifies to print the Help message.

-LOG | -LOGFILE <*filename*>
specifies the filename to use for spawner log output if you are not using the -LOGCONFIGLOC option. The -LOG option should not be used with the -LOGCONFIGLOC option. If both options are specified, then the -LOGCONFIGLOC option takes precedence.

You can specify the -DEBUG or -TRACE options with the -LOG <*filename*> option, so that the detailed spawner log messages are sent to a log file.

Example The following example specifies CNTSPAWN to start the SAS/CONNECT spawner and specifies that debug-level log messages are sent to a file named *unxspawner.log*.

```
cntspawn -start -debug -log unxspawner.log
```

-LOGCONFIGLOC <*filename*>
enables the SAS logging facility for SAS servers and names the location of the configuration file that is used by the SAS logging facility to create spawner log output. The configuration file is an XML file that specifies and configures loggers and appenders for the SAS/CONNECT spawner.

In a planned deployment, the SAS Deployment Wizard automatically creates an initial logging configuration file named `logconfig.xml` that you can modify as needed to adjust the spawner's logging configuration. This file is located in the *sas-installation-directory*`/Lev-`*n*`/ConnectSpawner/` directory on UNIX and the *sas-installation-directory*`\Lev-`*n*`\ConnectSpawner\` directory on Windows. The file contains the pattern layout for the messages that are generated and automatically directed to an output device, such as a console or a log file. Relevant log data for the Windows spawner might include the date and time, the log level, the thread ID, and the logger.

If you have a SAS Foundation installation, you can copy this file and customize it to your needs.

See "Sample Logging Configuration File" on page 423 for an example of a spawner log configuration file in the UNIX environment.

The file specification that defines the location of the XML configuration file must be a valid filename or a path and filename for your operating environment. If the path contains spaces, enclose the file specification in quotation marks.

Note: If LOGCONFIGLOC is specified, spawner messages are routed by default to the App.Connect.Spawner logger.

See "SAS Logging Facility" on page 421

Example "Sample Logging Configuration File" on page 423

-METAPASS <*password*>
specifies the password of the user who connects to the metadata server.

-METAPORT <*port*>
specifies the port to connect to on the metadata server.

-METASERVER <*host-name*> | <*IP-address*>'
specifies the name or IP address of the metadata server.

-METAUSER <*user-id*>
specifies the user ID of the user who connects to the metadata server.

-MGMTPORT <*port-number*> | <'*service-name*'>
enables you to specify the service name or number of the TCP/IP port that listens for operator connections. Operator connections are used to connect to the server to perform administrative tasks on the SAS/CONNECT spawner and server. For example, you can use the IOMOPERATE procedure with the spawner operator port to perform the following tasks:

- determine what servers are currently active through the SAS/CONNECT spawner and display information about the connected clients
- display information about the spawner
- display the number of connections made since the spawner started, the number of connections that are still active, and the number of sign-on failures to the server
- pause and re-start the spawner

For more information and a list of examples showing how to perform the tasks listed above, see "Use PROC IOMOPERATE to Manage the SAS/CONNECT Spawner" on page 322.

SAS automatically creates a spawner operator port and by default sets the port to 7541. Therefore, if you do not specify the -MGMTPORT option when starting the spawner, and port 7541 is already in use by another application, then the spawner will fail to start.

CAUTION
Do not specify the same port for both the -SERVICE option and the -MGMTPORT option. The spawner fails to start if both the -SERVICE and -MGMTPORT options are using the same port.

Default 7541

Range 1- 65535

Requirement A management port is required when setting up the SAS/CONNECT spawner. When starting the spawner, you must specify the -MGMTPORT option to set the spawner's operator port to a port other than the port used for the -SERVICE parameter. Otherwise, the spawner fails to start if both -SERVICE and -MGMTPORT are using the same port.

-NOCLEARTEXT
prevents sign-ons from clients that do not support user ID and password encryption. This option prevents clients that are running older releases (prior to SAS 6.09E and SAS 6.11 TS040, which do not support user ID and password encryption) from signing on to the spawner program. However, the default permits both encrypted and plaintext user IDs and passwords.

-NOINHERITANCE

disables socket inheritance. Socket inheritance enables SAS/CONNECT servers to use the socket connection that is established between the SAS/CONNECT client and the spawner. Socket inheritance saves resources and is easier to configure when clients connect to a server that is within a firewall. Socket inheritance is enabled by default.

-NOSCRIPT

prevents sign-on from clients that use scripts, and allows sign-on only from clients that do not use scripts.

-NOSCRIPT can be useful if you want to limit SAS start-up commands to the use of the -SASCMD option or to commands defined in metadata. Specifying -NOSCRIPT restricts clients from specifying additional options in SAS start-up commands or script files. When -NOSCRIPT is specified, either -SASCMD must also be specified or logical Connect Servers must be defined in metadata.

Note: In a SAS metadata server-based environment, if a scriptless server defined in metadata does not have a valid spawner SASCMD value, the logical server will be ignored.

-SASCMD | -CMD <*command*>

Windows

specifies the SAS command or a command file that invokes SAS when a client attempts to connect to a server.

- invoke SAS from a directory that is not the default location
- specify different SAS start-up command options
- execute other statements before invoking SAS

The -DMR, -COMAMID, -NOSPLASH, -ICON, and -NOTERMINAL options are supplied by default when you sign on using the SAS/CONNECT spawner.

In Windows, you can use either a batch file, which is signified by the .bat extension, or a command file, which is signified by the .cmd extension. Here is an example of a batch file:

```
cd !sasroot
sas.exe %*
```

The first line changes to the directory where the SAS executable is stored. The second line starts SAS. Add options as needed at this SAS start-up command.

UNIX

specifies the SAS command or a command file that is specific to the UNIX operating environment that starts a SAS session when you sign on without a script. If the client does not specify a script file at sign-on, the -SASCMD option must be specified when starting the spawner.

```
cntspawn -sascmd "/u/username/mystartup"
```

Here is a sample UNIX command file named **mystartup**:

```
#!/bin/ksh
#----------------------------------
# mystartup
#----------------------------------
```

```
. ~/.profile
sas -noterminal -nosyntaxcheck $*
#----------------------------
```

Note: The $* positional parameter enables you to specify additional SAS options when you invoke SAS.

z/OS
specifies a UNIX System Services (USS) shell script for starting a SAS session. You must use -SASCMD and a shell script if you do not specify a sign-on script in the client session using an RLINK fileref. The script interprets the command arguments and environment variables and builds a TSO command that invokes a SAS session. A sample USS shell script for starting a SAS session can be found in '**&previx.BAMISC(SPNCSHEL)**'.

For more information about using a shell script to start the z/OS spawner, see Figure 21.1 on page 364.

-SASDAEMONSERVICE *service-name*
specifies the service name or port number that the SAS/CONNECT server uses to listen for SAS child process connections. When socket inheritance is enabled, the SAS client and the SAS/CONNECT server communicate via this port. If you use a service, its name must be configured in the SERVICES file on the computer that the SAS/CONNECT server session runs on.

-SASSPAWNERCN <*name*>
specifies the name of the spawner definition to retrieve from the SAS Metadata Server.

If the -SASSPAWNERCN option is specified, you must either specify the -XMLCONFIGFILE option or you must specify the -METASERVER, -METAPORT, -METAUSER, and -METAPASS options. The -XMLCONFIGFILE option specifies the filename to use to get SAS Metadata Server access information. This file configures how the SAS/CONNECT Spawner connects to the SAS Metadata Server to retrieve its configuration information.

For details about generating a SAS/CONNECT spawner definition for the SAS Metadata Server, see the Help for the SAS/CONNECT spawner server type in the Server Manager of SAS Management Console.

-SERVICE <*port-number* | *service-name*>
specifies the service name or port number to use to listen for client connections.

The -SERVICE option values that are used to start the spawner determine what is used by the client to sign on.

In the following example, the spawner is started by specifying the *port-number* as the value of the -SERVICE option during spawner start-up:

```
"SAS-installation-directory\SASHome\SASFoundation\9.4\cntspawn.exe"
  -service 5020
```

The client can then sign on by specifying the explicit *port-number* in the SIGNON statement:

```
%let myHost=<spawner-host> 5020;
signon myHost;
```

Note If the -SERVICE option is not specified, the spawner listens on Telnet port (23).

See For information about the TCP/IP services file, see "Configure the TCP/IP Services File" on page 309.

-SHELL
specifies that the started SAS/CONNECT servers allows X commands.

Without specifying the -SHELL option to the spawner, X command processing is disabled by default. For details about running X commands from your SAS session, see *SAS Companion for Windows*.

-SSPI | -NOSSPI
identifies support for the Security Support Provider Interface for single sign-on connections to the spawner. If the client and the server run under Windows and if the client does not supply a user ID and password to the server, SSPI (Security Support Provider Interface) is used to perform client authentication. SSPI authentication is disabled by default. To enable SSPI authentication, you must specify -SSPI in the spawner start-up command. In versions prior to SAS 9.4, SSPI was enabled by default.

Default -NOSSPI

-TRACE | VERBOSE
turns on trace level output.

-XMLCONFIGFILE "*fully-qualified-path*"
specifies the filename to use to get SAS Metadata Server access information. A path that includes one or more spaces must be enclosed in quotation marks.

If -XMLCONFIGFILE is used, -SASSPAWNERCN must also be used.

Alias -OMRCONFIGFILE

Security Options

-ENCRYPTFIPS
specifies that the SAS/SECURE and TLS security services use FIPS 140-2 validated algorithms.

Note If the ENCRYPTFIPS option is specified on the command line or FIPS encryption is specified in metadata, then all encryption algorithms that are specified on the command line or in metadata must be either AES or SSL. Any other encryption algorithms result in errors.

See "ENCRYPTFIPS" in *Encryption in SAS*

Example The following example enables SSL and AES encryption.

```
"SAS-installation-directory\SASHome\SASFoundation\9.4\cntspawn.exe"
      -encryptfips
```

-METAENCRYPTALG *algorithm* | *NONE*
specifies the type of encryption algorithm to use when communicating with the metadata server. The following algorithms can be used: RC2, RC4, TripleDES, SAS Proprietary, and AES.

-METAENCRYPTLEVEL <*level*>
specifies the level of encryption when communicating with the metadata server.

-NETENCRYPT
specifies that network encryption is required.

See "NETENCRYPT" in *Encryption in SAS*

-NETENCRYPTALGORITHM
specifies the algorithm or algorithms to be used for encrypted client/server data transfers.

See "NETENCRYPTALGORITHM=" in *Encryption in SAS*

Note: If you are running SAS/CONNECT in a SAS Intelligence Platform environment using the SAS Metadata Server and configured encryption using SAS Management Console, you can specify only one encryption algorithm. For more information, see "ERROR: Cannot Negotiate Encryption Algorithm" in *Encryption in SAS* .

-NETENCRYPTKEYLEN
specifies the key length that is used by the encryption algorithm for encrypted client/server data transfers.

See "NETENCRYPTKEYLEN=" in *Encryption in SAS*

-SSLCALISTLOC <*filename*>
UNIX and z/OS only: specifies the name of the file that contains the list of trusted certificate authorities.

See "SSLCALISTLOC=" in *Encryption in SAS*

-SSLCERTISS <*issuer*>
Windows only: specifies the name of the issuer of the digital certificate that SSL should use.

See "SSLCERTISS=" in *Encryption in SAS*

-SSLCERTLOC <*filname*>
UNIX and z/OS only: specifies the name of the file that contains the public certificate to use for SSL.

See "SSLCERTLOC=" in *Encryption in SAS*

-SSLCERTSERIAL<*serial*>
Windows only: specifies the serial number of the digital certificate that SSL should use.

See "SSLCERTSERIAL=" in *Encryption in SAS*

-SSLCERTSUBJ <*subject*>
Windows only: specifies the subject name of the digital certificate that SSL should use.

See "SSLCERTSUBJ" in *Encryption in SAS*

-SSLCLIENTAUTH
specifies whether a server should perform client authentication.

See "SSLCLIENTAUTH" in *Encryption in SAS*

-SSLCRLCHECK
specifies whether a Certificate Revocation List (CRL) is checked when a digital certificate is validated.

See "SSLCRLCHECK" in *Encryption in SAS*

-SSLCRLLOC
UNIX and z/OS only: specifies the location of a Certificate Revocation List (CRL).

See "SSLCRLLOC=" in *Encryption in SAS*

-SSLPKCS12LOC
UNIX and z/OS only: specifies the location of the PKCS12 encoding package file.

See "SSLPKCS12LOC=" in *Encryption in SAS*

-SSLPKCS12PASS
UNIX and z/OS only: specifies the password that TLS requires to decrypt the PKCS12 file.

See "SSLPKCS12PASS=" in *Encryption in SAS*

-SSLPVTKEYLOC
UNIX and z/OS only: specifies the location of the private key that corresponds to the digital certificate.

See "SSLPVTKEYLOC=" in *Encryption in SAS*

-SSLPVTKEYPASS
UNIX and z/OS only: specifies the password that TLS requires for decrypting the private key.

See "SSLPVTKEYPASS=" in *Encryption in SAS*

Windows-only Service Options

Use the following service options to create, modify, and remove SAS/CONNECT spawner service definitions in the Windows operating environment:

-INSTALL <*options*>
causes an instance of a spawner to be installed as a Windows service. Each spawner instance is assigned a name by default in the following form:

```
SAS Connect Spawner
```

You can install each instance of the spawner by using the following command:

```
"sas-installation-directory\SASHome\SASFoundation\9.4\cntspawn.exe" -install
```

You can assign a different name to the spawner by using the -SERVICENAME option. If you try to install a second spawner without specifying the -SERVICENAME option, the attempt will fail and you will get an error.

The alias for the -INSTALL option is -I.

-INSTALLDEPENDENCIES *service-name*

service-name
specifies the name of the dependent Windows service that must be started before the spawner service can be started. This dependency can be viewed using the Microsoft Windows Services Manager plug-in (services.msc).

Valid in -INSTALL option statement

Alias -IDEP

-SERVICEDESCRIPTION *'service-description'*

'service-description'
specifies the description that you assign to the spawner that is installed and started as a Windows service using the -INSTALL option. The description can be viewed using the Microsoft Windows Services plug-in (services.msc). A specified spawner description cannot exceed 256 characters and must be enclosed in quotation marks if it contains one or more spaces. The following command installs a spawner named "**SAS spawner 5**" and specifies a description for the service:

```
"sas-installation-directory\SASHome\SASFoundation\9.4\cntspawn.exe"
 -install  -servicename "SAS spawner 5"
-servdesc "A SAS process that listens for
 requests to spawn SAS/Connect servers"
```

Valid in -INSTALL option statement only

Length 1-256

Alias -SERVDESC

Requirement must be enclosed in quotation marks if it contains one or more spaces

-SERVICEDIRECTORY *directory*

directory
specifies the directory in which to run the Windows service.

Valid in -INSTALL option statement only

Alias -SERVDIR

-SERVICENAME *'service-name'*

'service-name'
specifies the name that you assign to the spawner that is installed, or uninstalled, and started as a service in the Windows operating environment. A specified name overrides the default name that is automatically assigned when the -INSTALL option is used without specifying -SERVICENAME.

When you install a spawner without specifying -SERVICENAME, it is installed as **SAS Connect Spawner**. If you try to install a second spawner without specifying the -SERVICENAME option, the attempt will fail and you will get an error.

Valid in -INSTALL option statement only

Length 1-80

Alias -NAME

Requirement must be enclosed in quotation marks if it contains one or more spaces

Example The following example shows how to install an explicitly named spawner as a service:

```
"SAS-installation-directory\SASHome\SASFoundation\9.4\
cntspawn.exe"  -install -servicename "Doug's spawner"
```

-SERVICEPASS *password*

password
specifies the password for the user account that the spawner will run under as a service when you specify the -INSTALL option.

Alias -SERVPASS

See "Using TLS for Encryption of a SAS/CONNECT Windows Spawner: Example " in *Encryption in SAS*

-SERVICEUSER=*user-ID*

user-ID
specifies a user name that the Windows service will run under, when you also specify the -INSTALL option.

```
"sas-installation-directory\SASHome\SASFoundation\9.4\cntspawn " -install
```

Alias -SERVUSER

See -UNINSTALL on page 335

-UNINSTALL <-SERVICENAME *'service-name'*>
instructs the spawner to uninstall as a Windows service, which was previously installed and started by using the -INSTALL option.

If you used the -SERVICENAME option with the -INSTALL option to install a spawner, you can use the -SERVICENAME option with the -UNINSTALL option to identify the spawner to be removed. The following example shows how to uninstall an explicitly named Windows spawner by using the -UNINSTALL command. Use quotation marks around the pathname and command, as well as the spawner service name. Here is an example:

```
"SAS-installation-directory\SASHome\SASFoundation\9.4\cntspawn.exe"
 -uninstall -servicename "Doug's spawner"
```

Alias -DEINSTALL or -DI

Spawner Examples

Scripted Sign-on to a UNIX Spawner (Server)

From the UNIX node that the server runs on, specify the following command to start the spawner.

```
cntspawn -service spawn1 -mgmtport 5030
```

The -MGMTPORT option specifies the operator port to be used for administrative purposes. The -SERVICE option specifies the name of the service, **spawn1**, that listens for incoming server requests. The -service option can specify a defined TCP/IP service or a numeric port value. What is used when the spawner is started determines what will be used by the client. In the following example the-SERVICE option is used to specify the numeric port value of the service during spawner start-up:

```
cntspawn -service 5020 -mgmtport 5030
```

A user can then sign on using the same port-number in the SIGNON statement:

```
filename rlink '!sasroot\connect\saslink\tcpunx.scr';
signon rmthost.5020;
```

The -SERVICE option values used to start the spawner determine what will be used by the client to sign on.

As in the first example, the -MGMTPORT option specifies the operator port to be used for administrative purposes.

Scripted Sign-on to a UNIX Spawner (Client)

At a Windows client, the statements in the example below are used to sign on to the UNIX node **RMTHOST**. The script file, **tcpunx.scr**, which is assigned to the RLINK fileref, prompts the user at the client for the user ID and password. The user ID and password are needed to sign on to the UNIX server.

```
filename rlink '!sasroot\connect\saslink\tcpunx.scr';
signon rmthost.spawn1;
```

The server name (in this example, **RMTHOST**) is either the name of the UNIX node or a macro variable that contains the IP address or the name of the UNIX node that runs the spawner.

The SIGNON statement contains the ID of the server session, which is specified as a two-level name: the node name and the service name. A two-level name is needed when signing on to a UNIX node that runs a spawner.

Scriptless Sign-on to a Windows Spawner That Runs as a Service (Server)

The following command installs the spawner service on a Windows computer:

```
cntspawn -install
```

For this example, note the following:

- The spawner is being installed as a Windows service, but since the -SERVICE option is not used to specify the port number or name, the spawner will listen on the default Telnet port (23) and be named SAS Connect Spawner by default.
- Because a sign-on script is not being used and the -SASCMD= option is not specified letting the spawner know how to start SAS, the spawner will look for the SAS executable in the SAS installation directory. See "SAS/CONNECT Files" on page 407 for information about the names and location of default files related to SAS/CONNECT software.
- Since the -MGMTPORT is not specified, the operator port will default to 7541.

After the service is installed, it must be started before it can be used. You can start the service using either of the following:

- the NET START command

  ```
  net start "SAS Connect Spawner"
  ```

- the services applet
- a reboot of the computer
- the ConnectSpawner.bat script file command

Scriptless Sign-on to a Windows Spawner That Runs as a Service (Client)

From any client, the following statements connect to the spawner program by using the TCP/IP access method. The SIGNON statement specifies the ID of the server session REMNODE. This ID must be the name of the Windows computer or a macro variable that contains the IP address of the Windows computer that the spawner runs on. The user ID and password to the server are specified as options in the SIGNON statement. The value _PROMPT_ in the SIGNON statement causes SAS to prompt for the password.

```
signon remnode user=joeblack password=_prompt_;
```

For Windows users, if SSPI has been enabled, then you do not need to specify the user ID and password in the SIGNON statement. See "Use SSPI to Access a Secured Server" on page 387 for more information about SSPI.

Note: The password is displayed as Xs in the SAS log.

Encrypted Sign-on to a z/OS Spawner (Server)

The following z/OS command starts the z/OS spawner.

```
START SPAWNER
```

This command activates the started task procedure. SPAWNER is the name of the service that is defined in the started task procedure.

PARMFILE contains the options that start the spawner. For example:

```
-netencryptalgorithm rc2
-sascmd "/usr/local/bin/spawnsas.sh" -nosasuser -mgmtport=7451
```

1. The -MGMTPORT option specifies port 7451 as the port for operator connections.
2. -NETENCRYPTALGORITHM option – specifies that the spawner is started using the RC2 encryption algorithm.
3. -SASCMD option – specifies a UNIX System Services shell script that starts SAS. This command assumes that a shell script named `spawnsas.sh` is installed in `/usr/local/bin`.
4. -NOSASUSER - specifies that a user's SASUSER file should not be allocated. -NOSASUSER enables a client to sign on to a server multiple times using the same user ID and password.

Note: A sample started task procedure can be found in **'&prefix.BAMISC(SPNCCNTL)'**.

Encrypted Sign-on to a SAS/CONNECT Spawner (Client)

In the following Example, the client specifies user ID and password encryption by setting the RC2 encryption algorithm. In this example, the two-level name, which represents the node name and the service name, specifies the ID of the server session in the SIGNON statement. A two-level name is needed when signing on to a z/OS operating environment that runs a spawner. You must supply a valid user ID and password as values for the USER= and PASSWORD= options in the SIGNON statement.

```
options netencryptalgorithm=rc2;
signon rmthost.spawner user=joeblack password=born2run;
```

20

UNIX Operating Environment

Overview

Overview

What Is Covered

This section describes how to use SAS/CONNECT in a SAS Foundation environment for UNIX. If you are using SAS/CONNECT as part of a SAS Intelligence Platform Deployment (for example, SAS Business Intelligence Server or SAS Data Integration Server), refer to the SAS Intelligence Platform Documentation at http://support.sas.com/documentation/onlinedoc/intellplatform/tabs/admin94.html.

For a list of resources specifically related to using SAS/CONNECT in a SAS Intelligence Platform environment, see “SAS/CONNECT in a SAS Intelligence Platform Environment” on page 5. More detailed information describing the scope of this document can be found in the section “Document Scope” on page 4.

Types of Connections

This section contains information about how to use three types of connections that are available when using SAS/CONNECT software in a SAS Foundation environment:

- Spawner connections on page 345
- SASCMD connections on page 354
- Telnet connections on page 356

Regardless of the type of connection you are using, this document assumes that you have completed the configuration steps as outlined in the Configuration Guide for SAS 9.4 for Foundation on UNIX.

Network Requirements

Tasks

Before you begin using the SAS/CONNECT spawner on UNIX, you must complete the following steps:

- Verify that Base SAS and SAS/CONNECT are installed on both the client and the server.
- Complete the steps as outlined in Post-Installation Configuration for User Authentication and Identification in *Configuration Guide for SAS 9.4 Foundation for UNIX Environments*.
- Set environment variables for TCP/IP connections, as needed.
- Set SAS system options for TCP/IP, if needed.

Environment Variables

The following environment variables are available for configuring your TCP/IP connections. Environment variables can be set in a UNIX shell, in a configuration file, or in the OPTIONS statement with the SET system option. Examples in this section use the SET system option.

For more information about configuring environment variables in a UNIX environment, see "Defining Environmental Variables" in *SAS Companion for UNIX Environments*.

CONNECTWDWAIT

used to limit the possibility that a client session disconnect might orphan a runaway DMR mode session. To ensure the responsiveness of the spawner, SAS starts a 'watchdog' thread to monitor the connection. CONNECTWDWAIT can be specified on the CONNECT server session. The default interval is five seconds. If a disconnect occurs, CONNECTWDWAIT checks 18 times and then terminate the DMR thread (for a default elapsed time of 90 seconds). Setting the CONNECTWDWAIT value to zero means the process does not monitor the connection.

Defaults interval: 5 seconds

total elapsed time: 90 seconds

Examples In the following example, the option is set to 10, so the process waits for 180 seconds and then terminates the thread.

```
-set CONNECTWDWAIT 10
```

In the following example, the option is set to 0, so the process does not monitor the connection:

```
-set CONNECTWDWAIT 0
```

In the following MP CONNECT example, the option is set to 0, so the process does not monitor the connection.

```
signon t1 sascmd="!sascmd -set CONNECTWDWAIT 0";
```

TCP_POLL_INTERVAL

used to ensure responsiveness of SAS spawners and servers to various conditions outside of normal request processing. When idle, servers and spawners periodically awaken to check for requests. The interval in seconds for this check is governed by the TCP_POLL_INTERVAL environment variable. Generally, the default setting of 60 seconds should be acceptable. However, if you want to configure the interval, set it in the TKMVSENV file by specifying the TCP_POLL_INTERVAL variable. A value of zero means the server remains idle and awakens only for request processing.

Example In the following example, the option is set to 50, so the process checks every 50 seconds for a connection.

```
-set TCP_POLL_INTERVAL 50
```

TCPIPMCH

specifies the IBM TCP/IP stack name to set the stack affinity for z/OS systems that are running more than one TCP/IP stack. This environment variable alters default processing for TCP/IP initialization.

Example `-set TCPIPMCH <stack-name>`

TCPMSGLEN *n*

defines the size of the buffer (in bytes) that the TCP/IP access method uses for breaking up a message that it sends to or receives from the SAS/CONNECT application layer during a SAS/CONNECT session. The application layer uses a message size that is stored in the TBUFSIZE option that you can specify in the SIGNON statement or as a SAS option.

If TBUFSIZE is larger than TCPMSGLEN, the TCP/IP access method breaks the message into a buffer whose size is defined by TCPMSGLEN, and issues the number of send and receive messages that are necessary to complete the message transaction.

The value for TCPMSGLEN must be set at both the client and the server. If the values that are set for TCPMSGLEN at the client and at the server are different, the smaller value of the two is used during the SAS/CONNECT session. If the TCPMSGLEN environment variable is not set, SAS uses the TCP stack's default size and allows autotuning if implemented by the stack.

A value of zero means the server remains idle and awakens only for request processing. An idle server might be subject to S522 (Job Wait Time-out) abend. However, a spawner defined as an MVS started task or as a UNIX System Services daemon process should not be subject to idle wait termination.

Client Optional

Server Optional

See "TBUFSIZE=" on page 121

Example `-set TCPMSGLEN 65536`

TCPPROXYLIST

used to support HTTP_CONNECT so that SAS clients outside of the cloud can sign on to SAS/CONNECT spawners. By setting the TCPPROXYLIST environment variable, you can connect to different clouds from the same client.

If you provide a proxy list delimited by semicolons, the system parses the list and connects to the first proxy host or port. All subsequent proxies are sent an HTTP CONNECT request to create the tunnel on the final host or port.

Client Optional

Example
```
-set TCPPROXYLIST "http://machine-name-1:port-number;
http://machine-name-2:port-number"
```

CONNECTKEEPALIVE

Prevents a SAS/CONNECT client connection to the SAS/CONNECT server from being terminated.

Setting this environment variable in the server session prevents firewalls from terminating a connection between a client and server when there are long periods of inactivity on the connection. A keepalive packet is sent from a thread that is started by the server session for the specified number of seconds.

Example The value of 5 causes the keepalive packet to be sent every 5 seconds to prevent connection termination.
```
-set CONNECTKEEPALIVE 5
```

TCPBINDADDR<*IP_address*>

Specifies an IP address override for all listening ports to bind to.

It can be specified on a SAS/CONNECT client or SAS/CONNECT server invocation.

If the spawner uses its BINDADDR value, it passes this to the SAS/CONNECT server's environment by setting the TCPBINDADDR environment variable for the launched session. Then the launched SAS/CONNECT server uses this IP address to bind all of its listening ports to.

Default Listens on all IP addresses available for the host, if the variable is not specified.

Example
```
./sas -set TCPBINDADDR 10.20.16.48
```

When TCPBINDADDR is specified on the client using the grid SIGNON, the IP address is used by the grid server to connect back to the client instead of connecting to the hostname of the client.

In Platform LSF grids, LSF (Load Sharing Facility) copies the client environment variables to the execution host. The TCPBINDADDR environment variable should be unset in the grid server configuration to prevent the execution host trying to bind its listening ports to the client IP value.

Add the following to the grid usermods file:

For a Linux grid, the changes are added in ***comfig_dir*/LevX/*SASApplicationServerName*/GridServer/grid_usermods.cfg**

```
unset TCPBINDADDR
```

For a Windows grid, the changes are added in ***comfig_dir*\LevX*SASApplicationServerName*\GridServer\grid_usermods.cfg.cmd**

```
set "TCPBINDADDR="
```

System Options for TCP/IP

The following options can be used to control how SAS/CONNECT uses TCP/IP to establish connections:

TCPPORTFIRST= *port-number*
TCPPORTFIRST= *port-number*
restricts the range of TCP/IP ports that clients can use to remotely access servers. Within the range of 0 through 32767, assign a beginning value to TCPPORTFIRST and an ending value to TCPPORTLAST. To restrict the range of ports to only one port, set the values for TCPPORTFIRST and TCPPORTLAST to the same number. Consult with your network administrator for advice about these settings.

At the server, you can set TCPPORTFIRST and TCPPORTLAST in a SAS start-up command or in the configuration file.

In the example below, the server is restricted to the TCP/IP ports 4020 through 4050:

Server Optional

See “TCPPORTFIRST=” on page 125

Example
```
options tcpportfirst=4020;
options tcpportlast=4050;
```

TCPTN3270 (set at the client)
supports connections to z/OS servers that use the full-screen 3270 Telnet protocol. The script file TCPTSO32.SCR is provided.

For a list of sign-on scripts, see Table 23.3 on page 409.

You can set the TCPTN3270 option only in the SAS configuration file. If you do not set this option, the TCP/IP access method uses the Telnet line-mode protocol by default.

Client Optional

Example
```
-set TCPTN3270 1
```

Spawner Connections on UNIX

Set Up the Spawner on UNIX

Overview

This section contains the steps for setting up the SAS/CONNECT spawner in a SAS Foundation environment for UNIX.

If you have installed SAS/CONNECT as part of a planned deployment or as part of a SAS Intelligence Platform deployment, then most of this setup has been done for you by the SAS Deployment Wizard and you do not need to complete these tasks.

Information about configuring and managing the SAS/CONNECT spawner in a planned deployment can be found in the SAS Intelligence Platform Documentation. See “SAS/CONNECT in a SAS Intelligence Platform Environment” on page 5 for a list of resources for using SAS/CONNECT in the SAS Intelligence Platform environment.

Note: In this document, all references to the “spawner” or “spawner program” are intended to mean the *SAS/CONNECT* spawner or the *SAS/CONNECT* spawner program.

Tasks

- Verify that Base SAS and SAS/CONNECT are installed on both the client and the server.
- Start the spawner.
- Stop the spawner.

Network Security

If you are connecting to a UNIX server using the SAS/CONNECT spawner, SAS/CONNECT uses the default authentication mechanism to verify the user-ID and password of the client signing on.

See Post-Installation Configuration for User Authentication and Identification in *Configuration Guide for SAS 9.4 Foundation for UNIX Environments* for information

about configuring SAS to perform authentication and user validation in a UNIX operating environment.

Location of the SAS/CONNECT Spawner on UNIX

The SAS/CONNECT spawner executable file, cntspawn.exe, is located by default in the following directory:

SAS-installation-directory/utilities/bin/cntspawn

Start the Spawner

To start the SAS/CONNECT spawner on the UNIX server, specify the spawner invocation command as shown here:

```
cntspawn <options>
```

Example 1:

```
cntspawn -sascmd "/u/username/mystartup"
```

Example 2:

```
cntspawn -service 5020
```

In Example 1, the -SASCMD option is a spawner start-up option that is used to tell the spawner how to start SAS on the UNIX server. In Example 2, the -SERVICE option specifies the spawner's listening port.

For a complete list of other available spawner invocation options, see "Spawner Options" on page 326.

In a SAS Intelligence Platform deployment, or a planned deployment, you can use the following command to install the spawner on UNIX:

```
ConnectSpawner.sh  -install
```

This file and others are created by default when you install and configure SAS servers using the SAS Deployment Wizard. Then a spawner .sh file is created in the spawner's configuration directory. For more information, see Configuration Files for SAS Object Spawners and SAS/CONNECT Spawners in *SAS Intelligence Platform: System Administration Guide*.

Specify the Spawner Port or Service Name

To accept connection requests from SAS/CONNECT clients using TCP/IP, the spawner must be listening on a designated port. Therefore, a port number or TCP/IP service name is needed to be used as the spawner's listening port. The spawner's listening port is specified on spawner start-up using the -SERVICE option:

```
cntspawn -service <port-number> | <service-name>  <options>
```

Example:

```
cntspawn -service 7551
```

port–number is the port that the spawner listens on for client requests on the UNIX server.

service–name is the name of the spawner service

If you want to use a TCP/IP service name for the spawner's listening port instead of referring to the spawner using its explicit port number, you can set up a TCP/IP service name in the **services** file on the server. The TCP/IP service name corresponds to the spawner's listening port.

The TCP/IP service name is an arbitrary name that provides a convenient way for you to reference the spawner. The services file is a plain text file that provides a mapping between service names and their assigned ports. The **services** file is typically located in **/etc/services** directory on UNIX systems. Here are the steps for setting up the spawner TCP/IP service name:

1 Specify a *service–name* in the `cntspawn` spawner start-up command.

```
cntspawn -service mySpawner
```

2 Update the TCP/IP **services** file on the UNIX server by adding the name of the spawner service, its port number, and the communications protocol type (TCP).

For more information about the services file, see "Configure the TCP/IP Services File" on page 309.

In the following example, assume that the TCP/IP **services** file was updated to define a SAS/CONNECT spawner named `mySpawner` that is listening on port 5020. The following command starts the spawner and allows clients to connect using the TCP/IP service name, `mySpawner`, or the explicit port number.

```
cntspawn -service mySpawner -mgmtport 7555 -sascmd "/u/username/mystartup"
```

The -MGMTPORT spawner option specifies a spawner port for operator connections, to be used for administrative purposes.

Note: If the SERVICE option is not specified when the spawner is started, the spawner attempts to listen on Telnet port `23` and the service name is `SAS Connect Spawner` by default

Stop the Spawner

To end the spawner, type CTRL-C to kill the process.

Specify Encryption Options for Data Security

If you want to specify an encryption method other than the default SAS Proprietary Encryption, you can specify SAS system options for encryption on the spawner

start-up command. For example, you can specify the NETENCRYPTALGORITHM option on the spawner start-up command to specify various encryption algorithms such as RC2, DES, and SSL.

SAS proprietary encryption, which is enabled by default with SAS, provides a medium level of security and includes encryption for passwords used for communications in configuration files, encryption for login passwords, encryption for internal account passwords, and encryption of general traffic between clients and remote hosts. For more information about SAS Proprietary Encryption, see "SAS Proprietary Encryption" in *Encryption in SAS* . For more information about security options used with the SAS/CONNECT spawner, see "Security Options" on page 331.

In the following example, the **cntspawn** command starts a spawner named `unxspawn` and uses the -NETENCRYPTALGORITHM option to specify that data is encrypted using AES encryption:

```
cntspawn -service unxspawn -netencryptalgorithm aes
```

For more encryption examples, see the following sections in *Encryption in SAS*:

- "SAS/CONNECT Server on UNIX" in *Encryption in SAS*
- " Start-up of a UNIX Spawner on a SAS/CONNECT Server" in *Encryption in SAS*
- "Using TLS for Encryption of a SAS/CONNECT UNIX Spawner: Example " in *Encryption in SAS*

For a complete list of encryption options that can be used on the spawner start-up command, see "SAS System Options for Encryption" in *Encryption in SAS*.

Sign On to the SAS/CONNECT Spawner

Overview

This section contains the steps for signing on using the SAS/CONNECT spawner in a SAS Foundation environment.

Tasks

To sign on using the SAS/CONNECT spawner, complete the following steps:

1. Ensure that the spawner is running on the server.
2. Specify the server name and spawner port number or service name.
3. Sign on to the spawner using a script or sign on without a script.

Ensure That the Spawner Is Running on the Server

The server administrator must configure and start the spawner on the server before you can sign on using the spawner. The spawner cannot be started on the server by the client. For information about configuring and starting the SAS/CONNECT spawner on a UNIX server, see “Set Up the Spawner on UNIX” on page 345.

Specify the Server and the Spawner Port or Service Name

To sign on to a remote UNIX server that is running the SAS/CONNECT spawner, specify the name of the remote server, followed by the spawner port number or TCP/IP service name that is associated with the spawner port. You can specify the name of the server and the port number or service name in an OPTIONS statement or in a SIGNON statement. Here is the syntax for the OPTIONS statement:

```
OPTIONS REMOTE=node-name[.port-number] | [service-name];
```

Here is the syntax for the SIGNON statement:

```
SIGNON node-name[.port-number] | [service-name];
```

Example:

```
%let myNode=unixserv.us.123.com 5020;;
signon myNode;
```

- *node-name* is based on the remote UNIX server that you are connecting to. *node-name* must be a valid SAS name that is 1 to 8 characters in length and is one of the following:
  - the short name of the remote server that you are connecting to. The short host name must be defined in the HOSTS file in the client operating environment or in your Domain Name Server (DNS).
  - a macro variable that represents either the IP address of the host or the Fully Qualified Domain Name (FQDN) of the host that you are connecting to. Because FQDNs do not meet SAS naming requirements, you must assign the FQDN to a macro variable that meets these requirements. Here is an example:

    ```
    %let remhost=pc.rem.us.com;
    signon remhost.5050;
    ```

- *port-number* is the TCP/IP port that the spawner is listening on for client connections.
- *service-name* is the name associated with the port that the spawner is listening on.

You can specify the TCP/IP service name instead of the explicit port number to sign on to the spawner if the spawner has been set up to run on the server as a service.

To specify the spawner's TCP/IP service name when signing on, specify the name of the spawner service (*service-name*) in the SIGNON statement as follows:

```
%let myHost=pc.rem.us.com;
signon myHost.mySpawner;
```

The spawner service name and port number can be configured in the client's **services** file, but this is not a requirement. If you do configure the client's services file, the port number in the client **services** file must be identical to the spawner's listening port.

See "Specify the Spawner Port or Service Name" on page 346 for information about setting up the spawner to run as a service. For information about configuring the TCP/IP **services** file, see "Configure the TCP/IP Services File" on page 309.

Note: If you are using SAS/CONNECT with a metadata server, the spawner port number and service name can be defined in the metadata configuration file.

For more examples of signing on using the SAS/CONNECT spawner, see "Spawner Examples" on page 336.

Sign On Without a Script

If you are not using a sign-on script, then you must provide a user-ID and password to sign on to a secured server. Specify the USERNAME= and PASSWORD= options in the SIGNON statement.

Example:

```
%let rmthost=pc.rem.us.com;
options comamid=tcp;
signon rmthost.cntspwn1 user=_prompt_;
```

In the example, a client connects to a UNIX server by using a spawner without a script. In the SIGNON statement `rmthost` is the name of the server on which the spawner is running and `cntspwn1` is the name of the spawner service. The _PROMPT_ value in the USER= option causes a dialog box to appear so that a user-ID and a password can be provided.

In this scenario, assume that the server administrator has started the SAS/CONNECT spawner on a secured server using the -SERVICE option on the spawner start-up command as follows:

```
cntspawn -service cntspwn1
```

Since the spawner was started as `cntspwn1`, connecting clients must specify `cntspwn1` (or the associated port number) when signing on.

Sign On Using a Script

You can use a SAS/CONNECT sign-on script to sign on to a server that is running the spawner. The sign-on script is executed by the SIGNON statement and prompts for the client user-ID and password by default.

Note: If you do not use a sign-on script when connecting to a secured server, then you must supply a *user-ID* and *password* when signing on.

To sign on using a script, use the FILENAME statement with the default fileref, RLINK, to associate RLINK with the script that you want to use. Then, specify the SIGNON statement (without the fileref argument).

If you use the default SAS fileref, RLINK, you do not need to specify a fileref in the SIGNON statement. When the SIGNON executes, SAS automatically searches for a file that is defined using RLINK as the fileref and executes the script that is associated with it.

Example 1:

```
filename rlink "/misc/connect/tcpunix.scr";
signon;
```

Example 2:

```
filename rlink 'c:\Program Files\SASHome\SASFoundation\9.4\connect\saslink\
 tcptunix.scr';
options remote=rmtnode;
signon;
```

In the second example, a UNIX client executes the tcpunix.scr script to connect to a remote UNIX server. The FILENAME statement identifies the script file on the client machine that is used to sign on to the server. The script file was configured to contain a user-ID and a password that are valid on the server. The REMOTE= system option specifies the server rmtnode.

For more information about using RLINK in the FILENAME statement, see “Example 1: Use a FILENAME Statement for a Script File” on page 203.

Sample script files are provided with SAS/CONNECT for signing on and signing off. The script that you choose is based on the server that you are connecting to. The following table lists the names and locations of the scripts that are provided with SAS/CONNECT.

Table 20.1 *SAS/CONNECT Sign-on Scripts for TCP/IP*

Server/Operating System	Script Name	Location of Script File on Client
UNIX	tcpunix.scr	***!sasroot*/misc/connect/**
TSO under z/OS	tcptso.scr	**prefix.CTMISC**
TSO under z/OS, SAS 9 or later	tcptso9.scr	
z/OS (without TSO)	tcpmvs.scr	
z/OS (full-screen 3270 Telnet protocol)	tcptso32.scr	
Windows	tcpwin.scr	**‘*!sasroot*\connect\saslink\’**

Note: If the spawner was started using the -NOSCRIPT option, then you cannot use a script to sign on to the spawner. Assigning the fileref `RLINK`in a FILENAME statement is used only when signing on using a script.

For more information about using SAS/CONNECT script files and SAS/CONNECT script statements, see "SAS/CONNECT Sign-on Script Files" on page 408 and "Simple SAS/CONNECT Scripts for Sign On and Sign Off" on page 412.

For more information about the FILENAME statement, see "FILENAME" on page 201.

Specify Data Encryption Options for Sign-ons

Encryption is the process of transforming plaintext into a less readable form (called ciphertext) by using a mathematical process. The ciphertext is translated back to plaintext for anyone who can supply the appropriate key, which is necessary for decrypting (or unlocking) the ciphertext. SAS proprietary encryption, which is enabled by default with SAS, provides a medium level of encryption, including encryption for passwords used for communications in configuration files, passwords for login objects, login passwords, internal account passwords, and encryption of general traffic between clients and remote hosts. If you want to specify an encryption algorithm that is different from the default SAS Proprietary encryption, you can specify SAS system options for encryption.

For more information about security options used with the SAS/CONNECT spawner, see "Security Options" on page 331.

For more information about SAS Proprietary Encryption, see "SAS Proprietary Encryption" in *Encryption in SAS* .

In the following example, the NETENCRYPT option specifies the AES algorithm in the SAS OPTIONS statement:

```
options netencrypt=aes;
signon;
```

Here are more examples showing how to sign on using encryption:

- "Connection of a SAS/CONNECT Client to a UNIX Spawner" in *Encryption in SAS*
- "SAS/CONNECT Client on UNIX" in *Encryption in SAS*

For a list of SAS system options for Encryption, see "Spawner Options" on page 326.

Spawner Sign-on Examples

The following table contains examples of spawner sign-ons to a remote UNIX host. For all of the examples, assume that the spawner was configured and started on the remote host. The CLIENT column contains valid sign-on statements that can be used to sign on to the remote UNIX host machine.

Table 20.2 Spawner Sign-on Examples

CLIENT	Description
`%let r=pc.rem.us.com;` `signon r.7551;`	In this example, the client signs on using the host name **r** and the explicit port number 7551 that was used to start the spawner on the UNIX remote host. The fully qualified domain name of the remote host is assigned to the macro variable **r**. The explicit port number (7551) is specified with the host name in the SIGNON statement. Note that the REMOTE= or CONNECTREMOTE= option is implicit in the SIGNON statement.
`%let r=pc.rem.us.com 7551;` `signon r user=abc123 pass=******;`	In this example, the client signs on by specifying the remote host name, the user ID, and the password in the SIGNON statement. The fully qualified domain name of the remote host and the explicit port number (7551) are both stored in the macro variable **r**. The user ID and password are specified in the SIGNON statement.

`%let r=pc.rem.us.com 7551;`
`options remote=r;`
`signon user=abc123 pass=_prompt_;`
	In this example, the client signs on by specifying the remote host name in the OPTIONS statement. The fully qualified domain name of the remote host and the port number 7551 are both stored in the macro variable **r**. The user ID and password are specified in the SIGNON statement. The value for the PASSWORD option (_prompt_) causes the server to prompt the user for the password during the sign-on.

`%let r=pc.rem.us.com;`
`signon r._ _7551 user=abc123 pass=******;`
`libname myLib server=r._ _7551;`
	Again, the fully qualified domain name of the UNIX remote host is assigned to the macro variable **r**. The host name and port number are specified in the SIGNON statement using the "computer-name.__port-number" format. A libref is defined on the server using the LIBNAME SERVER= statement. When a LIBNAME statement is used in a sign-on, the double underscore format must be used to specify the server ID for both the SERVER= option in the LIBNAME statement and the SERVER= option in the SIGNON statement.

`%let r=10.5.55.14  7551;` `signon r user=abc123 pass=******;`	The IP address of the UNIX remote host and its port (7551) are assigned to the macro variable **r**. The host name (**r**), user ID, and password are specified in the SIGNON statement.

`%let r=pc.rem.us.com;`
`filename rlink 'C:\Program Files\`
`  SASHome\SASFoundation\9.4\`
`  connect\saslink\tcpunix.scr';`
`signon r;`
	This example shows a scripted spawner sign-on from a Windows client machine to a remote UNIX server. The fully qualified domain name of the UNIX host is assigned to the macro variable **r**. The FILENAME RLINK statement is used to specify the sign-on script. The FILENAME statement assigns the default fileref, RLINK, to the script file located in **C:\Program Files\SASHome\SASFoundation\9.4\connect\saslink** on the client machine.

CLIENT	Description
`filename rlink '..';` `signon mvshost._ _7551;`	This example shows a scripted spawner sign-on from a UNIX client machine to a remote UNIX server. The FILENAME RLINK statement specifies the parent of the current folder as the path for the script file. The UNIX host name and port number are specified in the SIGNON statement using the "computer-name.__port-number" format.

SASCMD Connections on UNIX

Sign On to the Same Multiprocessor Computer

Tasks

If your client computer is equipped with SMP, you can run one or more server sessions from the local session on the same computer. Here are the steps for creating one or more server sessions on your local computer:

1. Specify the server session.
2. Specify the SASCMD option to start SAS.
3. Sign on.

Specify the Name of the Server Session

You can specify the name of the server session using either of the following methods:

- in an OPTIONS statement using the REMOTE= system option:

  Example:

  ```
  options remote=session1;
  ```

  Note: REMOTE= is an alias for the CONNECTREMOTE= system option.

- in a SIGNON statement using the REMOTE= option:

  Example:

```
signon remote=session1;
```

Specifying the -REMOTE= option in the SIGNON statement is optional since the option is implied in the SIGNON statement:

```
signon session1;
```

Specify the SASCMD Option to Start SAS

Use the SASCMD option to specify the command to start a SAS session and any additional options that you want to use to start the session. The SASCMD option can be specified in an OPTIONS statement or in the SIGNON statement.

- Here is the syntax for the SASCMD= system option in the OPTIONS statement:

  ```
  SASCMD=<"SAS-command <SAS-system-options>" | "!SASCMD <SAS-system options"> >
  ```

  Example 1:

  ```
  options sascmd="sas -nosyntaxcheck -noterminal";
  ```

  Example 2:

  ```
  options sascmd="!sascmd -nosyntaxcheck -noterminal";
  ```

- Here is the syntax for the SASCMD= option in the SIGNON statement:

  ```
  sascmd="SAS-command" | "!sascmd" | "!SASCMDV" | "host-command-file"
  ```

  Example 1:

  ```
  signon sascmd="start_sas"  <options>;
  ```

- **!sascmd** tells SAS to start the server session using the same command that was used to start the local SAS session.
- **host-command-file** represents a sign-on script file that contains SAS start-up commands. These commands should be customized for your operating environment. For more information about the **host-command-file** value in the SASCMD option, see “"*SAS-command*"” on page 138.

For more information, see SASCMD= system option in the OPTIONS statement and SASCMD= option in the SIGNON statement.

Sign On

If you did not use the SIGNON statement as shown in the previous step to specify the server session and start-up command, then you must specify the SIGNON statement to complete the sign-on:

```
signon;
```

In the following example, the session name and the SAS start-up command are both specified in the OPTIONS statement, so a simple SIGNON statement without arguments can be used:

Example 1:

```
options process="session1" sascmd="start_sas";
signon;
```

Example 2:

In the following example, the value for the SASCMD= option is set in the OPTIONS statement and the name of the session is set in the SIGNON statement. The SASCMD= option specifies `sas` as the command to start the SAS server session.

```
options sascmd="sas";
signon session2;
```

The NOSYNTAXCHECK System Option

You can specify the NOSYNTAXCHECK system option when signing on to a server session on the same symmetric multiprocessing (SMP) computer that the client session is running on. This option is most useful when client and server sessions run on SMP hardware.

NOSYNTAXCHECK enables continuous processing of statements regardless of syntax error conditions. When SYNTAXCHECK is enabled, SAS uses additional resources to validate SAS statements while producing limited results.

For example, the first instance of a syntax error triggers syntax checking, which automatically sets the value of the OBS= system option to 0. Consequently, no observations can be created by subsequent SAS statements in the program. When executing SASCMD sign-ons or when executing debugged production programs that are unlikely to encounter errors, consider using the NOSYNTAXCHECK option.

Here is an example of a SAS invocation that runs on the same computer at which the client session runs:

```
signon smp sascmd="sas -nosyntaxcheck -noterminal";
```

Telnet Connections on UNIX

IMPORTANT With the June 2023 hot fix, Telnet is deprecated and it is recommended that you use SAS/CONNECT Spawner for client sign-ons. The -CLEARTEXT option has been deprecated and is no longer available. For more information, see SAS Note 70114.

Tasks

When signing on using a Telnet daemon, specify the server name and a sign-on script. The script file is executed by the SIGNON statement. By default, the script prompts for the user ID and password. Here are the tasks for signing on using a Telnet daemon:

1 Specify the server name.

2 Specify a sign-on script.

3 Sign on.

Specify the Server

The name of the server can be specified in the SIGNON statement or in the OPTIONS statement. Here is the syntax for specifying the server name in an OPTIONS statement:

```
OPTIONS remote=<computer-name>;
```

Here is the syntax for specifying the server name in the SIGNON statement:

```
SIGNON <computer-name>;
```

For more information about the REMOTE= system option, see CONNECTREMOTE= system option on page 108. For more information about the SIGNON statement, see SIGNON statement on page 127.

Specify a Sign-on Script

If you are signing on by using a script, you must specify the script that you want to use. The script file is executed by the SIGNON statement or command. By default, the script prompts for user ID and password. For more information, see “Sign On Using a Script” on page 350.

Sign On to the Server Session

Use the SIGNON statement to sign on to a remote server session.

In the following example, a UNIX client connects to a z/OS server using the TCP/IP access method. The FILENAME statement identifies the script file that you use to sign on to a server. The script file contains a prompt for a user ID and a password that are valid on the server. The COMAMID= option specifies the TCP/IP communications access method for connecting to the RMTNODE server, which is specified in the REMOTE= option.

```
filename rlink '!sasroot/misc/connect/tcptso.scr';
options comamid=tcp remote=rmtnode;
signon;
```

Note: REMOTE= is an alias for the CONNECTREMOTE= system option.

Examples

Example 1: Sign On to a z/OS Server from a UNIX Client

In this example, a client session that runs under UNIX uses the TCP/IP access method to connect to a z/OS server. The FILENAME statement specifies the script file, **tcptso.scr**, to use to sign on to the server. The script file contains a prompt for a user-ID and a password. The COMAMID= option specifies the TCP/IP communications access method for connecting to the server rmtnode, which is specified in the REMOTE= option.

UNIX example:

```
filename rlink '!sasroot/misc/connect/tcptso.scr';
options remote=rmtnode;
signon;
```

Note: REMOTE= is an alias for the CONNECTREMOTE= option.

Example 2: Start the SAS/CONNECT Spawner on UNIX

The following command starts the SAS/CONNECT spawner on a UNIX server:

```
cntspawn -service spawner -mgmtport 7555 -sascmd "/u/username/mystartup"
-netencryptalgorithm ssl
```

- The -SERVICE option specifies that the service named **spawner** listens for incoming connections. This example assumes that the SERVICES file on the server was updated to include an entry for spawner with an assigned port.
- The -MGMTPORT option specifies the port number for operator (administrative) connections.
- The -SASCMD option specifies the path to the **mystartup** command file, which starts SAS on the server.
- The NETENCRYPTALGORITHM option specifies the SSL encryption algorithm.

  Note: In order for the UNIX spawner to locate the appropriate server digital certificate for SSL encryption, you must specify the -SSLCERTLOC and -SSLPVTKEYLOC system options or the SSLPKCS12LOC and

SSLPKCS12PASS system options in the script that is specified by the -SASCMD option.

21

z/OS Operating Environment

Overview

What Is Covered

This section describes how to use SAS/CONNECT in a SAS Foundation environment for z/OS.

If you are using SAS/CONNECT as part of a SAS Intelligence Platform Deployment (for example, SAS Business Intelligence Server or SAS Data Integration Server), refer to the SAS Intelligence Platform Documentation at http://support.sas.com/documentation/onlinedoc/intellplatform/tabs/admin94.html.

For a list of resources specifically related to using SAS/CONNECT in a SAS Intelligence Platform environment, see “SAS/CONNECT in a SAS Intelligence Platform Environment” on page 5.

More detailed information describing the scope of this document can be found in the section “Document Scope” on page 4.

Types of Connections

This section contains information about how to use three types of connections that are available when using SAS/CONNECT software in a SAS Foundation environment:

- Spawner connections
- MP Connect (SASCMD) connections
- Telnet connections

Regardless of the type of connection you are using, this document assumes that you have completed the configuration steps as outlined in the *Configuration Guide for SAS 9.4 for Foundation on z/OS*.

Note: In this document, all references to the “spawner” or “spawner program” are intended to mean the *SAS/CONNECT* spawner or the *SAS/CONNECT* spawner program.

Spawner Connections on z/OS

Product Requirements

Before you begin using the SAS/CONNECT spawner on z/OS, you must complete the following steps:

- Configure SAS to use TCP/IP as outlined in System Configuration for Using SAS with TCP/IP in *Configuration Guide for SAS 9.4 Foundation for z/OS*.
- Install the SASCP TSO command as outlined in Implementing SAS TSO Support in *Configuration Guide for SAS 9.4 Foundation for z/OS*.
- Complete the post-installation configuration for SAS/CONNECT Software as outlined in Post-Installation Configuration for SAS/CONNECT Software in *Configuration Guide for SAS 9.4 Foundation for z /OS*. These steps include configuring SAS/CONNECT spawner security, setting up the started task procedure, and defining the SAS startup shell script.
- Ensure that the SAS SVC routine has been installed as outlined in Installing the SAS 9.4 SVC Routine in *Configuration Guide for SAS 9.4 Foundation for z/OS*.
- Specify spawner options and environment variables .
- Ensure that Base SAS and SAS/CONNECT software have been installed on both the client and the host machines.

Set Up the Spawner on z/OS

Steps

Here are the tasks that are associated with setting up the SAS/CONNECT spawner on z/OS:

- Complete the steps as outlined in the previous section.
- Configure TCP/IP ports.
- Specify a spawner TCP/IP service name, as needed.
- Specify other spawner invocation options, as needed.
- Start the spawner.
- Stop the spawner.

Spawner Components on z/OS

The SAS/CONNECT spawner runs as a z/OS started task and uses z/OS UNIX System Services (USS) to spawn each user's SAS/CONNECT server session. Each server session runs in a `BPXAS` address space, executing the UNIX USS `/bin/tso` command to run the SAS REXX start-up command. Therefore, to start the SAS/CONNECT spawner, you need to first configure the started task procedure in the system PROCLIB library. The spawner uses TCP/IP socket services for inter-process communications.

A sample USS script file can be found in `'&prefix.BAMISC(SPNCSHEL)'`.

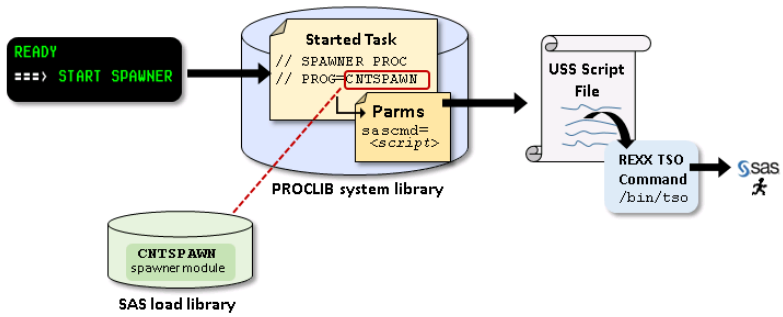
Note: The `/bin/tso` command mentioned above is used by default in the UNIX shell script. If you need to run authorized commands in SAS 9.3 and later releases, use the `/bin/tsocmd` command instead. See Usage Note 54530 for information about setting the `/bin/tsocmd` command in the UNIX shell script.

If you have satisfied the product requirements as outlined in the section Post-Installation Configuration for SAS/CONNECT Software in the *Configuration Guide for SAS 9.4 Foundation for z /OS*, then you should have already configured the spawner started task procedure and PARMS options. A sample started task procedure can be found in `'&prefix.BAMISC(SPNCCNTL)'`.

The SAS/CONNECT spawner module is named CNTSPAWN and it is located in the `SAS load library`. This library is allocated by default to the STEPLIB DD in the spawner started task procedure. Alternatively, the library can be installed in LPA or LINKLIST. The spawner started task requires a PARMS file for specifying spawner invocation options. A sample PARMS file can be found in `'&prefix.BAMISC(SPNCPARM)'`.

Here is a conceptual diagram showing the components discussed here and their relationship to one another:

Figure 21.1 *Components of the SAS/CONNECT Spawner on z/OS*



The -SERVICE and the -MGMTPORT options must be specified in the spawner's started task procedure or in the PARMS file for the spawner to start correctly.

Configure TCP/IP Ports

- -SERVICES Option (Listening Port)

  To accept connection requests from SAS/CONNECT clients using TCP/IP, the spawner must be listening on a designated port. Therefore, a port or TCP/IP service name is needed to be used as the spawner's listening port. The spawner's listening port is specified using the -SERVICE option in the started task procedure JCL. Alternatively, the -SERVICE PARM can be moved to the started task procedure's PARMS file. Here is the syntax for the -SERVICE option:

  -SERVICE <*port-number*> | <*service-name*>

- -MGMTPORT Option (Operator Port)

  On z/OS, a SAS/CONNECT spawner TCP/IP operator port is defined on the remote host by default. The operator port is set by default to 7541. If you want the operator port to use a port other than the default port, then change the -MGMTPORT PARM in the started task procedure JCL. Alternatively, the -MGMTPORT PARM can be moved to the PARMS file. Here is the syntax for the -MGMTPORT option:

  -MGMTPORT <*port-number*> | <*service-name*>

Note: On z/OS, you can specify a *service-name* for the MGMTPORT option starting with the third maintenance release of SAS 9.4. For releases prior to this, you can specify the *port-number* only for the -MGMTPORT option.

For more information about the -SERVICE option, see -SERVICE on page 330. For more information about the -MGMTPORT option, see -MGMTPORT on page 328.

Specify a Spawner TCP/IP Service Name (Optional)

Note: If you intend to start the spawner using an explicit port number as described in "Configure TCP/IP Ports" on page 365, you do not need to perform this step.

If you want to use a TCP/IP service name for the spawner's listening port instead of referring to the spawner using its explicit port number, you can set up a TCP/IP service name that corresponds to the spawner's listening port. The TCP/IP service name is an arbitrary name that provides a convenient way for users to reference the spawner. The services file is a plain text file that maps service names to port numbers. Here are the steps for setting up the spawner TCP/IP service name:

1 Add the TCP/IP service name to the -SERVICE option in the started task procedure JCL or PARMS file.

```
-SERVICE mySpawner
```

2 Update the TCP/IP services file on the remote host by adding the name of the TCP/IP service, its port number, and the protocol type (TCP). For more information about updating the TCP/IP service file, see “Configure the TCP/IP Services File” on page 309.

Specify Other Spawner Options (Optional)

SAS/CONNECT spawner start-up options are specified in the PARMS file, which is used by the spawner started task procedure to start the spawner. The following table lists some of the more commonly used spawner options. For a complete list of spawner invocation options, see “Spawner Options” on page 326.

Table 21.1 *Commonly Used SAS/CONNECT Spawner Options on z/OS*

-MGMTPORT	(required in SAS 9.4 and later releases) specifies the TCP/IP service name or port number that the spawner listens on for operator connections. Operator connections are used for spawner administrative tasks. In SAS 9.4 and later releases, if you do not specify the operator port using the -MGMTPORT option, SAS defaults to using TCP port 7541. If that port is in use, the SAS/CONNECT spawner fails to start. On z/OS, the ability to use a *service-name* rather than the explicit port number when defining the spawner operator port applies to SAS versions later than the third maintenance release of SAS 9.4.
-SERVICE	specifies the TCP/IP port number or TCP/IP service name that the spawner listens on for client requests. For more information about the -SERVICE option, see “Spawner Options” on page 326.
-NOSCRIPT	prevents sign-ons from clients that use scripts, and allows sign-ons only from clients that do not use scripts.
-NOCLEARTEXT	prevents sign-ons from clients that do not support user ID and password encryption. SAS releases prior to SAS 6.09E (MVS) and SAS 6.12 (Windows and UNIX) do not support user ID and password encryption. If the spawner is started with the NOCLEARTEXT option specified, then clients running these versions of SAS will not be able to connect to the SAS/CONNECT spawner.
-SASCMD	specifies a UNIX System Services shell script that starts SAS.
-TRACE | VERBOSE	used with the -LOG | -LOGFILE option to specify the level of logging for the SAS/CONNECT spawner.

	Specifying either option causes the log output to be more detailed.
-NETENCRYPTALGORITHM	specifies the type of encryption to use if you do not want to use the default SAS Proprietary encryption. SAS proprietary encryption is enabled by default with SAS and provides a medium level of encryption. For more information about security options that are used with the SAS/CONNECT spawner, see "Security Options" on page 331.

Note: SAS/SECURE is now part of Base SAS.

For more information about SAS Proprietary Encryption, see "SAS Proprietary Encryption" in *Encryption in SAS* .

Note: In SAS 9.4 and later releases, the -INHERITANCE option is no longer a valid spawner option. If -INHERITANCE is set in your **PARMS** file and you are running SAS 9.4 or later release, you will get an error when starting the SAS/CONNECT spawner procedure. You should remove the -INHERITANCE from your **PARMS** file and restart your SAS/CONNECT spawner.

Start the Spawner

If you have configured the started task procedure, use the following syntax to start the spawner on z/OS:

```
START <started-task-procedure>
```

Example:

```
START SPAWNER
```

This command starts the **SPAWNER** started task procedure. The spawner continues to run until it is stopped.

Stop the Spawner

To stop the spawner, enter the following command:

```
STOP <started-task-procedure>
```

Example:

```
STOP SPAWNER
```

Specify Encryption for Spawner Start-up

If you want to specify an encryption method other than the default SAS Proprietary Encryption, you can specify SAS system options for encryption on the spawner start-up command. For example, you can specify the NETENCRYPTALGORITHM option in the PARMS file to specify various encryption algorithms such as RC2, DES, and SSL, to name just a few.

SAS proprietary encryption, which is enabled by default with SAS, provides a medium level of security and includes encryption for passwords used for communications in configuration files, encryption for login passwords, encryption for internal account passwords, and encryption of general traffic between clients and remote hosts. For more information about SAS Proprietary Encryption, see "SAS Proprietary Encryption" in *Encryption in SAS*. For more information about security options used with the SAS/CONNECT spawner, see "Security Options" on page 331.

Sign On to the Spawner

Task List

Here are the steps for signing on to the SAS/CONNECT spawner that is running on a z/OS host:

1. Ensure that the spawner is running on the remote host.
2. Specify TCP/IP as the access method (applies to z/OS clients only).
3. Specify the host name and TCP/IP port number or service name.
4. Specify the sign-on script (optional).
5. Specify a user-ID and password for the sign-on.
6. Submit the sign-on code.

Ensure That the Spawner Is Running on the Remote Host

Before you can access the spawner, the spawner program must be running on the host machine.

If you are a SAS administrator and need information about setting up and starting the SAS/CONNECT spawner on z/OS, see "Set Up the Spawner on z/OS" on page 363.

Specify the Access Method

TCP/IP is the default communications access method for the Windows and UNIX operating environments. On z/OS, XMS is the default access method. Therefore, if you are signing on from a z/OS client session, you need to specify TCP/IP as the access method using the COMAMID system option as follows:

```
options comamid=tcp;
```

Sign On Using the Host Name and Spawner Port Number

To sign on to the spawner, you must know the name of the remote host computer that you are connecting to and the TCP/IP port number (or service name) that the spawner is listening on. The value of the port number that you specify in the sign-on statement should be identical to the port number that was specified as the spawner's listening port on the remote host.

Note: Instead of specifying the explicit TCP/IP port number, you can specify the port's TCP/IP service name. For information about specifying the TCP/IP service name on the client host, see Specify the TCP/IP Service Name on page 370.

When signing on, the name of the remote host and spawner port number (or TCP/IP service name) can be specified in an OPTIONS statement or in a SIGNON statement. The syntax for the OPTIONS statement is as follows:

```
OPTIONS REMOTE=node-name[.port-number | service-name>]  <more options>;
```

The syntax for the SIGNON statement is as follows:

```
SIGNON node-name[.port-number | service-name >]  <more options>;
```

port-number is the TCP/IP port on the remote host that the spawner is listening on for client requests.

service-name is the TCP/IP service name on the client machine.

node-name is based on the host that you are connecting to. *node-name* must be a valid SAS name that is 1 to 8 characters in length and is one of the following:

- the short name of the remote host that you are connecting to. This short host name must be defined in the client's HOSTS file where the short name is mapped to the IP address of the host machine or defined in your Domain Name Server (DNS).
- a macro variable that represents either the IP address of the host or the Fully Qualified Domain Name (FQDN) of the host that you are connecting to. Because FQDNs do not meet SAS naming requirements, you must assign the FQDN to a macro variable that meets these requirements. Here is an example:

```
%let remhost=zos.rem.us.com;
signon remhost.5050;
```

SAS evaluates the *node-name* in the following manner:

1. If *node-name* is a macro variable, the value of the macro variable is passed to the operating environment's `getnameinfo()` function.
2. If *node-name* is not a macro variable or the value of the macro variable does not produce a valid value, *node-name* is passed to the `getnameinfo()` function.
3. If `getnameinfo()` fails to resolve *node-name* to an IP address, an error message is returned and the sign-on fails.

Note: The order in which the `getnameinfo()` function calls the DNS or searches the HOSTS file to resolve *node-name* varies based on the operating environment's implementation of TCP/IP.

In the following example, the FQDN of the remote host is `zos.rem.us.com`, which is not a valid SAS name. Therefore, the macro variable `remhost` is assigned to the host name using the %LET macro statement. The host name that is specified in the SIGNON statement uses the macro variable `remhost` to sign on. The port number 5050 is also specified in the SIGNON statement:

```
%let remhost=zos.rem.us.com;
signon remhost.5050;
```

You can also include the port number in the definition of the macro variable. Here is an example:

```
%let remhost=zos.rem.us.com 5050;
signon remhost;
```

Specify the TCP/IP Service Name (Optional)

You can use a TCP/IP service name for spawner signons rather than referring to the spawner using its port number. The TCP/IP services file on the host machine must be configured to map the port-number to the TCP/IP service name. You can update the TCP/IP services file on the client machine to map the TCP/IP service name to the spawner's port number but this is not a requirement.

If you do configure the client's services file, the port number that is mapped to the TCP service name must match the port number that was used for the spawner start-up on the remote host. Here is the syntax:

```
SIGNON node-name.service-name;
```

The TCP/IP services file is a plain text file that maps TCP/IP service names to port numbers. You can use a text editor to add an entry to the services file. The entry must include the name of the TCP/IP service, the port number that is associated with that service, and the communications protocol being used (TCP).

For more information about updating the TCP/IP service file, see "Configure the TCP/IP Services File" on page 309.

Note: Remember, this step is optional. You can always start the spawner or sign on to it using the spawner's explicit port number.

Specify a Sign-on Script (Optional)

If you want to override the default launch command that is specified in the PARMS file by the -SASCMD PARM, you can use a sign-on script to sign on to the remote host. If you are signing on using a script, you must specify the script that you want to use and you must customize the script to match the logon process to your remote host.

The script file is executed by the SIGNON statement. By default, the script prompts for user ID and password. If you do not use a sign-on script and the spawner is running secured, you must supply a user ID and password when signing on using the spawner.

To use one of the sample script files that are supplied with SAS/CONNECT for signing on and signing off, assign the default fileref RLINK to the appropriate script file. The script that you choose is based on the remote host that you are connecting to. On z/OS client machines, the sample scripts are installed in `&prefix.CTMISC`. You can determine the location of script files on UNIX and Windows client machines by executing the following OPTIONS procedure in your local UNIX or Windows SAS session:

```
proc options option=sasscript;
```

To specify a script, use the FILENAME statement as follows:

```
FILENAME RLINK 'script-file-location';
```

script-name specifies the appropriate script file for the server. The following table lists SIGNON scripts that are supplied with SAS software.

Note: Script filenames on z/OS do not have a **.scr** extension.

Table 21.2 *SAS/CONNECT Sign-on Scripts for Using TCP/IP under z/OS*

Type	Name of Script File
TSO under z/OS	**tcptso.scr**
TSO under z/OS, SAS 9 or later	**tcptso9.scr**
z/OS (direct logon)	**tcpmvs.scr**
z/OS (using full-screen 3270 Telnet protocol)	**tcptso32.scr**
UNIX	**tcpunx.scr**
Windows	**tcpwin.scr**

Note: You cannot sign on to a spawner using a script file if the spawner was started using the -NOSCRIPT PARM. If the FILENAME RLINK script file is allocated to the

client SAS session, it will be used automatically and can cause conflict with PARMS options specified in the SIGNON statement, such as USER= and PASS=.

Specify a User ID and Password

If you are signing on to the spawner without using a script and the spawner is running secured, then you must submit a password and user ID in the SIGNON statement to connect to the remote z/OS host:

```
SIGNON USER=user-ID | _PROMPT_ [ PASSWORD=password | _PROMPT_ ];
```

Submit the Sign-on Code

In the following example, a client connects to a remote host through a spawner without using a script file. In the SIGNON statement, **rmthost.spawner** specifies the node, **rmthost**, and the TCP/IP service name, **spawner**.

Specifying USER=_PROMPT_ causes a logon dialog box to appear so that a user ID and a password can be provided.

```
options comamid=tcp;
signon rmthost.spawner user=_prompt_;
```

Sign-on Examples

The following table contains examples of valid spawner signons. For all of the examples, assume that the spawner has been configured and started on the server. The CLIENT column contains valid sign-on statements that can be used to sign on to the remote host.

Table 21.3 *Spawner Sign-on Examples*

CLIENT	Description
`%let r=zos.rem.us.com;` `signon r.7551;`	The fully qualified domain name of the remote host is assigned to the macro variable **r**. The services file is not configured on the remote host. The client signs on using the host name **r** and the explicit port number 7551 that was used to start the spawner on the remote host. **Note:** Because an explicit port number is specified, this example is not valid if you are signing on and using remote library services (RLS). To use RLS and sign-on using an explicit port number, you must specify the port number using the "computer-name.__port-number" format.

CLIENT	Description
`%let r=zos.rem.us.com 7551;` `signon r user=abc123 pass=****;`	The fully qualified domain name of the remote host and the port number 7551 are both assigned to the macro variable **r**. The host name, the user ID, and the password are specified in the SIGNON statement. **Note:** Because an explicit port number is specified, this example is not valid if you are signing on and using remote library services (RLS). To use RLS and sign-on using an explicit port number, you must specify the port number using the "computer-name.__port-number" format.

`%let r=zos.rem.us.com 7551;`
`options remote=r;`
`signon user=abc123 pass=******;`
	The fully qualified domain name of the remote host and the port 7551 are both assigned to the macro variable **r**. The host name is specified in the OPTIONS statement. The user ID and password are specified in the SIGNON statement.
	Note: Because an explicit port number is specified, this example is not valid if you are signing on and using remote library services (RLS). To use RLS and sign-on using an explicit port number, you must specify the port number using the "computer-name.__port-number" format.

`%let r=zos.rem.us.com;`
`signon r._ _7551`
`user=abc123 pass=******;`
	The fully qualified domain name of the remote host is assigned to the macro variable **r**. The host name and port number are specified in the SIGNON statement using the "computer-name.__port-number" format.

`%let r=10.5.55.14 7551;` `signon r user=abc123 pass=******;`	The IP address of the remote host and port 7551 are assigned to the macro variable **r**. The host name, user ID, and password are specified in the SIGNON statement. **Note:** Because an explicit port number is specified, this example is not valid if you are signing on and using remote library services (RLS). To use RLS and sign-on using an explicit port number, you must specify the port number using the "computer-name.__port-number" format.
`signon zoshost._ _7551` `user=abc123 pass=******;`	The fully qualified domain name of the host and the port number are specified in the SIGNON statement using the "computer-name.__port-number" format.

`%let r=zos.rem.us.com;`
`filename rlink '  .';`
`signon r;`
	The fully qualified domain name of the host is assigned to the macro variable **r**. The FILENAME RLINK statement specifies the current folder as the path for the script file.

`filename rlink '  ..';` `signon mvshost._ _7551;`	The FILENAME RLINK statement specifies the parent of the current folder as the path for the script file. The host name and port number are specified in the SIGNON statement using the "computer-name.__port-number" format.

Enable Encryption for Spawner Sign-ons

If you want to specify an encryption method other than the default SAS Proprietary Encryption, you can specify SAS system options for encryption in the PARMS file. For example, you can specify the NETENCRYPTALGORITHM option in the PARMS file to specify various encryption algorithms such as RC2, DES, and SSL to name just a few.

SAS proprietary encryption, which is enabled by default with SAS, provides a medium level of security and includes encryption for passwords used for communications in configuration files, encryption for login passwords, encryption for internal account passwords, and encryption of general traffic between clients and remote hosts. For more information about SAS Proprietary Encryption, see “SAS Proprietary Encryption” in *Encryption in SAS*. For more information about security options used with the SAS/CONNECT spawner, see “Security Options” on page 331.

In the following example, the client specifies user ID and password encryption by setting the RC2 encryption algorithm. The two-level name, which represents the node name and the service name, specifies the ID of the remote host session in the SIGNON statement. A two-level name is needed when signing on to a z/OS operating environment that runs a spawner. You must supply a valid user ID and password as values for the USER= and PASSWORD= options in the SIGNON statement.

```
options netencryptalgorithm=rc2;
signon rmthost.spawner user=joeblack password=born2run;
```

For details about encryption services, see the *Encryption in SAS*, located in the Base SAS Help and Documentation. For a complete list of SAS encryption options, see “SAS System Options for Encryption” in *Encryption in SAS*.

MP Connections on z/OS

Overview

Multi-Process Connect (or MP Connect) is a feature of SAS/CONNECT software that enables you to run multiple SAS sessions in parallel on the same multiprocessing computer. MP Connect sessions, therefore, run on symmetric multiprocessing (SMP) systems that support multiprocessing within a single operating environment.

On z/OS, the XMS access method enables SAS/CONNECT to run these types of sessions in parallel within a single z/OS environment. To use the XMS access method with SAS/CONNECT, both the client and remote host sessions must run on the same computer (or node).

In addition, to run XMS on z/OS, you must satisfy the network requirements that are outlined in "Product Requirements for the XMS Access Method" on page 375 *Configuration Guide for SAS Foundation for z /OS* before you can use XMS to complete SMP connections.

Product Requirements for the XMS Access Method

Tasks

Before you begin using XMS with SAS/CONNECT, you must perform the following tasks:

- Complete the steps for configuring your system for the Cross-Memory Access Method as outlined in the section System Configuration for the Cross-Memory Access Method in *Configuration Guide for SAS 9.4 Foundation for z /OS*.
- Complete the steps for configuring SAS/CONNECT for connections on the same multiprocessor machine as outlined in the section SAS/CONNECT to the Same Multi-Process Machine on z/OS in *Configuration Guide for SAS 9.4 Foundation for z /OS*.
- Ensure that the SAS SVC routine has been installed as outlined in Installing the SAS 9.4 SVC Routine in *Configuration Guide for SAS 9.4 Foundation for z /OS*.
- Verify that Base SAS software and SAS/CONNECT software have been installed on both the client and the remote host machines.

Sign On Using MP Connect

Tasks

1 Specify XMS as the communications access method by specifying the COMAMID in an OPTIONS statement:

```
options comamid=xms;
```

Note: Since XMS is the default communications access method for SMP connections to a z/OS operating environment, you do not have to explicitly specify the access method.

2 Specify the name of the remote host session in the SIGNON statement:

```
SIGNON session-ID;
```

You can also specify the remote host session in an OPTIONS statement:

```
OPTIONS PROCESS=session-ID;
```

Note: PROCESS= is an alias for the CONNECTREMOTE= system option.

session-ID must be a valid SAS name that is 1 to 8 characters in length, and is the name that you assign to the remote host session on the multiprocessor computer.

3 Start SAS by specifying the SASCMD= system option. The SASCMD= option can be specified in either an OPTIONS statement or in a SIGNON statement.

- Specify the SASCMD= system option in an OPTIONS statement:

```
options sascmd=": nonumber";
```

  SASCMD specifies a colon that is followed by any SAS invocation options.

- Specify the SASCMD= option in a SIGNON statement:

```
signon sascmd=":ls=256"
```

Note: The -DMR option is automatically appended to the command. If *! SASCMD* is specified, SAS/CONNECT starts SAS on the remote host by using the same command that was used to start SAS for the current (parent) session.

Note: To execute additional commands before starting SAS, you can write a script that contains the SAS start-up commands that are appropriate for the operating environment. Specify this script as the value in the SASCMD= option.

Examples for Signing On Using MP Connect

Example 1

In the following example, XMS is the access method, SAS1 is the name of the host session, and the MEMSIZE= option is used when starting SAS on a multiprocessor computer.

```
options comamid=xms;
signon sas1 sascmd=":sort=6";
```

Example 2:

In the following example, OPTIONS statements set the values for the COMAMID= , the SASCMD=, and the PROCESS= options. The SASCMD= option is a non-blank value that causes the same CLIST that was used to start the client session to be used to start the host session. The PROCESS= option identifies the host session on the same multiprocessor computer. Because the SASCMD= and the PROCESS= options are defined, only a simple SIGNON statement is needed.

```
options comamid= xms sascmd="abc";options process=sas1;signon;
```

Here is a summary of the language elements used to sign on with MP Connect:

- COMAMID – specifies the access method. For more information, see “COMAMID=” on page 101 .

- PROCESS – specifies the host session name (alias CONNECTREMOTE). For more information, see For information about the PROCESS= system option, see “CONNECTREMOTE=” on page 108.
- SIGNON – signs you on to the host session. For more information, see “SIGNON” on page 127.
- SASCMD – starts SAS. For more information, see “SASCMD=” on page 113.

Telnet Connections on z/OS

IMPORTANT With the January 2024 hot fix, Telnet is deprecated and it is recommended that you use SAS/CONNECT Spawner for client sign-ons. The -CLEARTEXT option has been deprecated and is no longer available. For more information, see SAS Note 70114.

Product Requirements for Telnet Connections

Task List

- Complete the steps for setting up SAS Foundation for TCP/IP on z/OS as outlined in System Configuration for Using SAS with TCP/IP in *Configuration Guide for SAS 9.4 for Foundation on z/OS*.
- Verify that Base SAS software and SAS/CONNECT software have been installed on both the client and the host machines.
- Verify that Telnet is enabled on both the local and remote hosts.
- Specify the name of the remote host.
- Specify a sign-on script.

Sign On Using a Telnet Connection

Specify the Host

The name of the remote host can be specified in an OPTIONS statement:

```
OPTIONS REMOTE=<node-name>;
```

Or, you can specify it directly in the SIGNON statement or command:

```
SIGNON <node-name>;
```

Specify a Sign-on Script

To specify a sign-on script, use the FILENAME statement with the fileref, RLINK followed by the SIGNON statement. The SIGNON statement initiates the script in the RLINK file.

```
filename rlink '!sasroot/misc/connect/tcptso9.scr';
options comamid=tcp remote=rmtnode;
signon;
```

You must specify a sign-on script when connecting using Telnet.

Several sign-on scripts that enable you to connect and log on to the remote host are shipped with SAS/CONNECT software. Table 21.2 on page 371 describes the purpose and location of the sample sign-on scripts that are shipped with SAS.

For information about creating and customizing SAS/CONNECT script files, see “SAS/CONNECT Sign-on Script Files” on page 408.

Note: The script files that are shipped with SAS must be customized to match the logon process to your remote host. Connecting to the remote host via Telnet outside of SAS is recommended to see the necessary screens and messages that need to be handled by the script.

Example: Sign On to a Remote z/OS Host Session

In the following example, you specify the statements at a z/OS client to use the TCP/IP access method to connect to a remote host. The FILENAME statement identifies the script file that you use to sign on to the remote z/OS host. The script file contains a prompt for a user ID and a password that are valid on the remote host. The COMAMID= option specifies the TCP/IP communications access method for connecting to the remote host **rmtnode**, which is specified in the REMOTE= option.

```
filename rlink 'prefix.CTMISC(tcptso9.scr)';
options comamid=tcp remote=rmtnode;
signon;
```

Environment Variables

SAS environment variables on z/OS can be specified in the SAS data set TKMVSENV. If you use SAS environment variables, you must store them in the TKMVSENV file allocated for the appropriate SAS session. Alternatively, you can use the SET= system option. The use of these variables depends on the type of connection that you are establishing. For example, the TCP_POLL_INTERVAL variable is set for spawner connections only and the TCPMSGLEN variable can be set for either Telnet or spawner connections. The Valid In section in each of the following environment variables provides this information.

Here is a list of SAS environment variables that can be specified in the TKMVSENV file:

CONNECTWDWAIT

used to limit the possibility that a client session disconnect might orphan a runaway DMR mode session. To ensure the responsiveness of the spawner, SAS starts a 'watchdog' thread to monitor the connection. The default interval is five seconds. If a disconnect occurs, CONNECTWDWAIT will check 18 times and then terminate the DMR thread (for a default elapsed time of 90 seconds). Setting the CONNECTWDWAIT value to zero means the process will not monitor the connection.

Valid in spawner connections

Defaults interval: 5 seconds

total elapsed time: 90 seconds

Examples In the following example, the option is set to 10, so the process will wait 180 seconds then terminate the thread.

```
set CONNECTWDWAIT=10
```

In the following example, the option is set to 0, so the process will not monitor the connection:

```
set CONNECTWDWAIT=0
```

TCP_POLL_INTERVAL

used to ensure responsiveness of SAS spawners and hosts to various conditions outside of normal request processing. When idle, hosts and spawners periodically awaken to check for requests. The interval in seconds for this check is governed by the TCP_POLL_INTERVAL environment variable. Generally, the default setting of 60 seconds should be acceptable. However, if you want to configure the interval, set it in the TKMVSENV file by specifying the TCP_POLL_INTERVAL variable.

A value of zero means the host will remain idle and awaken only for request processing. An idle host might be subject to S522 (Job Wait Time-out) abend. However, a spawner defined as a z/OS started task or as a UNIX System Services daemon process should not be subject to idle wait termination.

Valid in spawner connections

Example In the following example, the option is set to 50, so the process checks every 50 seconds for a connection.

```
set TCP_POLL_INTERVAL=50
```

TCPIPMCH

TCPIPMCH specifies the IBM TCP/IP stack name to set the stack affinity for z/OS systems that are running more than one TCP/IP stack. This environment variable alters default processing for TCP/IP initialization.

Valid in Telnet and spawner connections

Example `set TCPIPMCH=<stack-name>`

TCPMSGLEN <*n*>

defines the size of the buffer (in bytes) that the TCP/IP access method uses for breaking up a message that it sends to or receives from the SAS/CONNECT application layer during a SAS/CONNECT session. The application layer uses a message size that is stored in the TBUFSIZE option that you can specify in the SIGNON statement or as a SAS option.

If TBUFSIZE is larger than TCPMSGLEN, the TCP/IP access method breaks the message into a buffer whose size is defined by TCPMSGLEN, and issues the number of send and receive messages that are necessary to complete the message transaction.

The value for TCPMSGLEN must be set at both the client and the host. If the values that are set for TCPMSGLEN at the client and at the host are different, the smaller value of the two is used during the SAS/CONNECT session. If the TCPMSGLEN environment variable is not set, SAS uses the TCP stack's default size and allows autotuning if implemented by the stack.

A value of zero means the host will remain idle and awaken only for request processing. An idle host might be subject to S522 (Job Wait Time-out) abend. However, a spawner defined as an MVS started task or as a UNIX System Services daemon process should not be subject to idle wait termination.

Client Optional

Server Optional

Valid in Telnet and spawner connections

See "TBUFSIZE=" on page 121

Example

```
set TCPMSGLEN=65536
```

TCPPROXYLIST

used to support HTTP_CONNECT so that SAS clients outside of the cloud can sign on to SAS/CONNECT spawners. By setting the TCPPROXYLIST environment variable, you can connect to different clouds from the same client.

If you provide a proxy list delimited by semicolons, the system parses the list and connects to the first proxy host or port. All subsequent proxies are sent an HTTP CONNECT request to create the tunnel on the final host or port.

Client Optional

Example

```
set TCPPROXYLIST="http://machine-name-1:port-number;
http://machine-name-2:port-number"
```

CONNECTKEEPALIVE

Prevents a SAS/CONNECT client connection to the SAS/CONNECT server from being terminated.

Setting this environment variable in the server session prevents firewalls from terminating a connection between a client and server when there are long periods of inactivity on the connection. A keepalive packet is sent from a thread that is started by the server session for the specified number of seconds.

Example The value of 5 causes the keepalive packet to be sent every 5 seconds to prevent connection termination.

```
-set CONNECTKEEPALIVE=5
```

TCPTN3270

supports connections to a z/OS host that uses the full-screen 3270 Telnet protocol. The sample sign-on script, `tcptso32.scr`, can be used as a template for these types of Telnet connections. See Table 23.3 on page 409 for a complete list of sign-on scripts.

You can set the TCPTN3270 variable only in the SAS CLIST. To set the TCPTN3270 variable, follow these steps:

- Set the TCPTN3270 CLIST variable at the client.
- Add TCPTN3270(1) to the SAS CLIST.

If you do not set this variable, the TCP/IP access method uses the Telnet line mode protocol by default.

Valid in Telnet connections

Note TCPTN3270 is set at the client.

See "Invoking SAS under TSO: the SAS CLIST" in *SAS Companion for z/OS*.

TKOPT_SVCNO=

specifies the number that corresponds to the type of SVC routine that was installed. The default value is 109 for the ESR SVC 109. If you are using the "user" SVC instead of the ESR SVC, this option should be set to the SVC number that was defined during installation of the SVC routine (SVC routines 200–255). To set this variable, use the following syntax:

```
set TKOPT_SVCNO=200
```

See Installing the SAS 9.4 SVC Routine in *Configuration Guide for SAS Foundation for z /OS* for information about installing the SVC routine.

Valid in spawner connections

'&prefix.TKMVSENV(TKMVSENV)' data set

Used by XMS access method

TKOPT_SVCR15=

species the routing code that was chosen when the SAS 9.4 SVC routine was installed into your operating system. This option applies only if the SAS 9.4 SVC was installed as an ESR Type 4 SVC. The default is 4 for compatibility with previous releases of SAS 9.4. To set this variable, use the following syntax:

```
set TKOPT_SVCR15=
```

See Installing the SAS 9.4 SVC Routine in *Configuration Guide for SAS Foundation for z /OS* for information about installing the SVC routine.

Valid in spawner connections

‘&prefix.TKMVSENV(TKMVSENV)’ data set

Used by XMS access method

22

Windows Operating Environment

Overview

What Is Covered

This section describes how to use SAS/CONNECT in a SAS Foundation environment for Windows. If you are using SAS/CONNECT as part of a SAS Intelligence Platform Deployment (for example, SAS Business Intelligence Server or SAS Data Integration Server), refer to the SAS Intelligence Platform Documentation at http://support.sas.com/documentation/onlinedoc/intellplatform/tabs/admin94.html.

For a list of resources specifically related to using SAS/CONNECT in a SAS Intelligence Platform environment, see “SAS/CONNECT in a SAS Intelligence Platform Environment” on page 5. More detailed information describing the scope of this document can be found in the section “Document Scope” on page 4.

Types of Connections

This section contains information about how to use three types of connections that are available when using SAS/CONNECT software in a SAS Foundation environment:

- Spawner connections on page 390
- SASCMD connections on page 400
- Telnet connections on page 402

Regardless of the type of connection you are using, this document assumes that you have completed the configuration steps as outlined in the Configuration Guide for SAS 9.4 for Foundation on Windows or the Configuration Guide for SAS 9.4 for Foundation on Windows for x64 (depending on your environment).

Network Requirements

Tasks

Before you begin using the SAS/CONNECT spawner on Windows, you must complete the following steps:

- Verify that Base SAS and SAS/CONNECT are installed on both the client and the server.
- Complete the steps as outlined in SAS/CONNECT Configuration in *Configuration Guide for SAS 9.4 Foundation for Windows Environments*.
- If running the SAS/CONNECT server secured, familiarize yourself with the two methods for authenticating clients. See simulated logon method on page 386 and SSPI on page 387 for more information.
- Set the appropriate “SAS/CONNECT Options for TCP/IP”, if needed.

Note: In this document, all references to the “spawner” or “spawner program” are intended to mean the *SAS/CONNECT* spawner or the *SAS/CONNECT* spawner program.

User Context in a Secured Server

Definition

User context is the identifying credentials of the client who is attempting to access a secured server. Identifying credentials include the user ID, password, and file access permissions. Users can specify their own user context or a different user context when accessing a server.

Users can specify different user contexts when logging on to a server by using someone else's user ID and password. Supplying someone else's user ID and password gives permission to users to access files that they might otherwise be denied access to. A system administrator's user ID and password is an example of a different user context that might be specified. Such a context does not belong to the user but can be granted to the user for access to specific files.

Access a Secured Server Using Your Own Context

To access a secured server by using your own user context, specify your user ID and password.

Note: Note: If SSPI is available, you must specify the user ID explicitly in a sign-on script or as an option in the SIGNON statement for SAS/CONNECT or in the LIBNAME statement for SAS/SHARE. For details, see “Use SSPI to Access a Secured Server” on page 387.

Access a Server Using a Different Context

To access a server by using a different context, specify the appropriate user ID and password.

Note: If SSPI is available, you must specify the user ID explicitly in a sign-on script or as an option in the SIGNON statement for SAS/CONNECT or in the LIBNAME statement for SAS/SHARE. For details, see “Use SSPI to Access a Secured Server” on page 387.

Server Security Using Client Authentication

Security for a SAS/CONNECT server's resources can be enforced only by authenticating the identity of the user who runs the client session that is accessing the server session.

Authentication is the act of verifying the identity of the user who is attempting to access a machine—that is, the machine that either the client session or the server session runs on. Authentication is performed so that a machine can use the identity information to make decisions about the user's authority to access protected resources. Under Windows, the user ID, password, and access permissions make up a user context.

Resources on a SAS/CONNECT server are considered to be protected when both of the following conditions are met:

- The server requires that the client provide its identity.
- The client presents an identity that is successfully authenticated.

After the client's identity is authenticated, the client is given the appropriate permissions to access the server's resources.

Under Windows, two methods are available for authenticating a client's identity:

- simulated logon method
- SSPI

The Simulated Logon Method to Access a Secured Server

Overview

The simulated logon method is the most commonly used method of authentication and is available in all SAS supported operating environments. In a simulated logon, the client provides a user ID and password that are checked by the server.

You use a simulated logon in the following situations:

- The client or the server (or both) does not run on a Windows machine.
- The user who runs the client machine is not a trusted user at the server machine.
- The user who runs the client machine wants to log on by using a different user context.

Requirements for Using Simulated Logon with SAS/CONNECT

To authenticate user credentials (user ID and password) of SAS/CONNECT or SAS/SHARE clients, the administrator of the computers that the SAS/CONNECT client and server sessions or the SAS/SHARE client and server sessions run on must assign the appropriate rights to users.

Here are the requirements for SAS/CONNECT and SAS/SHARE:

- assignment of the “Log on as batch job” right to users in client sessions that access SAS/CONNECT server sessions.
- assignment of the “Act as part of the operating system” right to users who start SAS/SHARE servers or SAS/CONNECT spawners.

Here are the requirements for SAS/CONNECT only:

- assignment of the “Increase quotas” right to users who start a SAS/CONNECT spawner.
- assignment of the “Replace a process level token” right to users who start a SAS/CONNECT spawner.

Note: Because the SAS/CONNECT spawner usually runs as a service under the LocalSystem account, these permissions are already set by default and user rights do not need to be changed.

Use SSPI to Access a Secured Server

Overview of SSPI

Security Support Provider Interface (SSPI), also referred to as Integrated Windows Authentication (IWA), enables transparent authentication for connections between Windows computers. Users that are members of a trusted domain are authenticated automatically, and user context information is transferred to the server.

Windows attempts to use SSPI for authentication whenever a user ID is not explicitly supplied.

SSPI is available only when the client and the server sessions both run on Windows computers, and the user who runs the client computer is a member of a domain that is trusted at the server computer.

SSPI Requirement for SAS/CONNECT

In versions prior to SAS 9.4, SSPI is enabled by default. To disable it, specify -NOSSPI on the spawner command. In SAS 9.4 and later, -SSPI is not enabled by default, and you must specify -SSPI on the spawner start-up command to enable it.

Note: If you used the SAS Deployment Wizard to configure and deploy SAS in a planned deployment, the -SSPI option is automatically added to the `ConnectSpawner.bat` and `ConnectSpawner.sh` script files. To disable it, edit the script files and remove the -SSPI option from the script, or use the -NOSSPI option when you sign on.

SAS/CONNECT Options for TCP/IP

TCPPORTFIRST= <*port-number*>
TCPPORTLAST= <*port-number*>

restrict the range of TCP/IP ports that clients can use to remotely access servers. Within the range of 0 through 32767, assign a beginning value to TCPPORTFIRST and an ending value to TCPPORTLAST. To restrict the range of ports to only one port, set the values for TCPPORTFIRST and TCPPORTLAST to the same number. Consult with your network administrator for advice about these settings.

At the server, you can set TCPPORTFIRST and TCPPORTLAST in a SAS start-up command or in the configuration file.

In the example below, the server is restricted to the TCP/IP ports 4020 through 4050.

Server Optional

Valid in SAS start-up command or SAS configuration file

See “TCPPORTFIRST=” on page 125

Example

```
options tcpportfirst=4020;
options tcpportlast=4050;
```

SAS/CONNECT Environment Variables for TCP/IP

The following environment variables are available for configuring your TCP/IP connections. You can define an environment variable in a configuration file using the SET system option, using the Windows SET command, or using the System dialog box. Examples in this section use the SET system option in the configuration file.

For more information about configuring environment variables in a Windows environment, see "Using External Files under Windows" in *SAS Companion for Windows*.

CONNECTWDWAIT

used to limit the possibility that a client session disconnect might orphan a runaway DMR mode session. To ensure the responsiveness of the spawner, SAS starts a 'watchdog' thread to monitor the connection. The default interval is five seconds. If a disconnect occurs, CONNECTWDWAIT will check 18 times and then terminate the DMR thread (for a default elapsed time of 90 seconds). Setting the CONNECTWDWAIT value to zero means the process will not monitor the connection.

Defaults interval: 5 seconds

total elapsed time: 90 seconds

Examples In the following example, the option is set to 10, so the process will wait 180 seconds then terminate the thread.

```
-set CONNECTWDWAIT 10
```

In the following example, the option is set to 0, so the process will not monitor the connection:

```
-set CONNECTWDWAIT 0
```

TCP_POLL_INTERVAL

used to ensure responsiveness of SAS spawners and servers to various conditions outside of normal request processing. When idle, servers and spawners periodically awaken to check for requests. The interval in seconds for this check is governed by the TCP_POLL_INTERVAL environment variable. Generally, the default setting of 60 seconds should be acceptable. However, if you want to configure the interval, set it in the TKMVSENV file by specifying the TCP_POLL_INTERVAL variable. A value of zero means the server will remain idle and awaken only for request processing.

Example In the following example, the option is set to 50, so the process will check every 50 seconds for a connection.

```
-set TCP_POLL_INTERVAL 50
```

TCPMSGLEN *n*

defines the size of the buffer (in bytes) that the TCP/IP access method uses for breaking up a message that it sends to or receives from the SAS/CONNECT application layer during a SAS/CONNECT session. The application layer uses a message size that is stored in the TBUFSIZE option that you can specify in the SIGNON statement or as a SAS option.

If TBUFSIZE is larger than TCPMSGLEN, the TCP/IP access method breaks the message into a buffer whose size is defined by TCPMSGLEN, and issues the number of send and receive messages that are necessary to complete the message transaction.

The value for TCPMSGLEN must be set at both the client and the server. If the values that are set for TCPMSGLEN at the client and at the server are different, the smaller value of the two is used during the SAS/CONNECT session. If the TCPMSGLEN environment variable is not set, SAS uses the TCP stack's default size and allows autotuning if implemented by the stack.

A value of zero means the server will remain idle and awaken only for request processing. An idle server might be subject to S522 (Job Wait Time-out) abend.

However, a spawner defined as an MVS started task or as a UNIX System Services daemon process should not be subject to idle wait termination.

Client Optional

Server Optional

See “TBUFSIZE=” on page 121

Example
```
-set TCPMSGLEN 65536
```

TCPPROXYLIST

used to support HTTP_CONNECT so that SAS clients outside of the cloud can sign on to SAS/CONNECT spawners. By setting the TCPPROXYLIST environment variable, you can connect to different clouds from the same client.

If you provide a proxy list delimited by semicolons, the system parses the list and connects to the first proxy host or port. All subsequent proxies are sent an HTTP CONNECT request to create the tunnel on the final host or port.

Client Optional

Example
```
-set TCPPROXYLIST "http://machine-name-1:port;
http://machine-name-2:port"
```

CONNECTKEEPALIVE

Prevents a SAS/CONNECT client connection to the SAS/CONNECT server from being terminated.

Setting this environment variable in the server session prevents firewalls from terminating a connection between a client and server when there are long periods of inactivity on the connection. A keepalive packet is sent from a thread that is started by the server session for the specified number of seconds.

Example The value of 5 causes the keepalive packet to be sent every 5 seconds to prevent connection termination.
```
-set CONNECTKEEPALIVE=5
```

Spawner Connections on Windows

Set Up the Spawner on Windows

Overview

This section contains the steps for setting up the SAS/CONNECT spawner in a SAS Foundation environment.

If you have installed SAS/CONNECT as part of a planned deployment or as part of a SAS Intelligence Platform deployment, then most of this setup has been done for you by the SAS Deployment Wizard and you do not need to complete these tasks.

Information about configuring and managing the SAS/CONNECT spawner in a planned deployment of SAS can be found in the SAS Intelligence Platform Documentation. See "SAS/CONNECT in a SAS Intelligence Platform Environment" on page 5 for a list of resources for using SAS/CONNECT in the SAS Intelligence Platform environment.

Note: In this document, all references to the "spawner" or "spawner program" are intended to mean the *SAS/CONNECT* spawner or the *SAS/CONNECT* spawner program.

Task List

1. Verify that Base SAS and SAS/CONNECT are installed on both the client and the server.
2. Assign user rights.
3. Install the spawner as a Windows service.
4. Start the spawner.
5. Stop the spawner.
6. Uninstall the spawner service.

Assign User Rights for a Server That Is Running Secured

By default, when the SAS/CONNECT spawner is installed as a Windows service, it runs under the LocalSystem User ID, which has all the required User Rights for running the SAS/CONNECT spawner. Those user rights requirements are as follows:

- "act as part of the operating system" for the user who runs the spawner
- "increase quotas" for the user who runs the spawner
- "replace process level tokens" for the user who runs the spawner
- "log on as batch job" for all clients who need to connect to the server

If you do not install the SAS/CONNECT spawner as a Windows service (that is, if you run it from your system prompt), the Windows User ID that is used to start the SAS/CONNECT spawner must be the local Administrator of the machine and must have the following User Rights:

- "bypass traverse checking" (the default is everyone)

- “log on locally”(the default is everyone) for all clients who need to connect to the server
- “act as part of the operating system” for the user who runs the spawner
- “increase quotas” or the user who runs the spawner
- “replace a process level token” or the user who runs the spawner

The Windows user ID specified at sign-on needs only the User Right “log on as a batch job.”

Install the Spawner on Windows

The SAS/CONNECT spawner executable file, **cntspawn.exe**, is located in the directory in which SAS is installed at your site or on your computer. The spawner executable is installed by default in the following location:

```
SASHome\SASFoundation\9.4\cntspawn.exe
```

Here is an example of the spawner location in a typical SAS Foundation installation on Windows:

```
C:\Program Files\SASHome\SASFoundation\9.4\cntspawn.exe
```

The spawner start-up command, **cntspawn.exe**, is copied by default to the **\SASFoundation\9.4** directory when you install SAS, but it is not installed by default. You must install the SAS/CONNECT spawner in the Windows operating environment before you can start the spawner.

To install the SAS/CONNECT spawner on a Windows server, specify `cntspawn -install` from the Windows command line, using the full pathname to the executable in quotation marks, as shown here:

```
"<SAS-installation-directory>\SASFoundation\9.4\cntspawn.exe" -install
```

After running this command, Windows installs the SAS/CONNECT spawner as a service and assigns the default name **SAS Connect Spawner**, to the spawner service. You can assign a different name to the spawner service by using the -SERVICENAME option with the -INSTALL option on the CNTSPAWN command.

```
cntspawn -install -servicename "mySpawner"
```

For more information about using the Windows -SERVICENAME option, see -SERVICENAME.

In a SAS Intelligence Platform deployment (planned deployment), you can use the following command to install the spawner on Windows:

```
ConnectSpawner.bat  -install
```

This file and others are created by default when you install and configure SAS servers using the SAS Deployment Wizard. Then a spawner batch file, ConnectSpawner.bat, is created in the spawner’s configuration directory. For more information, see Configuration Files for SAS Object Spawners and SAS/CONNECT Spawners in *SAS Intelligence Platform: System Administration Guide*.

Start the Spawner

Once you have installed the spawner, you can start it using one of the following methods:

- specify the NET START command with the spawner service name:

```
NET START "SAS Connect Spawner"
```

- use the Windows Services Manager plug-in to start the spawner service.

Specify the Spawner Port or Service Name

The spawner runs as a TCP/IP service. By default, if you do not specify a port number for the spawner, it listens on port 23. If you want the spawner to run on a port other than port 23, then you must specify the -SERVICE option when you install the spawner. Here is the syntax for specifying the TCP/IP port number or service name:

```
cntspawn -install -service <port-number> | <service-name>  <options>
```

Example:

```
"c:\Program Files\SASHome\SASFoundation\9.4\cntspawn.exe" -install -service 5110
```

If you want to use a TCP/IP service name for the spawner's listening port instead of referring to the spawner using its explicit port number, then you must configure the spawner service name in the TCP/IP **services** file on the Windows server.

The TCP/IP services file is a configuration file that maps port numbers to named services. This mapping allows programs to access ports by name as well as by port number.

In Windows, the default location of the TCP/IP services file is **c:\Windows\System32\drivers\etc**. To configure the spawner service, use a text editor to add the connection service name, the port number, and the communications protocol to the services file. Here are the steps for setting up the spawner to run as a service on Windows:

1. Specify a service name in the `cntspawn` spawner start-up command.

```
cntspawn -service mySpawner
```

2. Update the TCP/IP **services** file on the Windows server by adding the name of the spawner service, its port number, and the communications protocol type (TCP).

If you do not specify a port number and port 23 is in use, then the spawner will not install and you get the following error message: "The spawner cannot listen on port 23."

See "Configure the TCP/IP Services File" on page 309 for information about configuring the spawner as a TCP/IP service.

Stop the Spawner

Once the spawner is installed and started, it can be stopped using any one of the following methods:

- specify the NET STOP command with the spawner service name:

```
NET STOP "SAS Connect Spawner"
```

- use the Windows Services Manager plug-in stop the spawner service.
- type CTRL-C or double-click in the upper left corner of the window in which the spawner is running.

Uninstall the Spawner Service

You can uninstall the spawner by specifying the -UNINSTALL option with the CNTSPAWN command. Specify the full pathname to the CNTSPAWN executable in quotation marks, as shown here:

```
"SASHome\SASFoundation\9.4\cntspawn.exe" -uninstall
```

If you used the -SERVICENAME option with the -INSTALL option when you installed the spawner to explicitly name the spawner service, then you must use the -SERVICENAME option with the -UNINSTALL option to uninstall it. Use quotation marks around the full pathname and around the spawner service name.

```
"SAS-installation-directory\SASFoundation\9.4\cntspawn.exe" -uninstall
-servicename "mySpawner"
```

For more information about using the Windows -UNINSTALL option, see -UNINSTALL.

Specify Encryption Options for Data Security

You can use options in the spawner invocation for encrypting SAS client/server data transfers.

If an encryption service is set up in your environment, specify the SAS system options that are appropriate for the encryption service that is being used. SAS system options for encryption can be set when you start the SAS/CONNECT spawner or when you sign on to a remote server session.

See “Start-Up of a Windows Spawner on a Single-User SAS/CONNECT Server” in *Encryption in SAS* for an example of specifying encryption options when starting the SAS/CONNECT spawner on Windows.

For a list of security options to specify on the spawner invocation command, see “Security Options” on page 331.

For more information about SAS encryption in general, see *Encryption in SAS*.

Example

The following command starts the SAS/CONNECT spawner on Windows:

```
cntspawn -service spawner -mgmtport 7555
   -sascmd "c:\Temp\start_sas.bat" -netencryptalgorithm aes
```

- The -SERVICE option specifies that the service named **spawner** listens for incoming connections. This example assumes that the SERVICES file on the server was updated to include an entry for `spawner` with an assigned port.
- The -MGMTPORT option specifies the port number for operator (administrative) connections.
- The -SASCMD option specifies the path to the **start_sas** command file, which starts SAS on the server.
- The -NETENCRYPTALGORITHM option specifies the AES encryption algorithm.

Sign On Using the Spawner

Tasks

1. Ensure that the spawner is running on the server.
2. Specify the server and the spawner service name or port number.
3. Sign on using a script or sign on without a script.
4. Specify a user ID and password.

Ensure That the Spawner Is Running on the Server

Before you can access the spawner, the system administrator for the machine that the spawner runs on must start the spawner. The spawner program on the server cannot be started by the client.

Specify the Server and the Spawner Service Name or Port Number

If you are running the SAS/CONNECT spawner as a service, you can sign on by specifying the spawner's service name in the OPTIONS statement. Verify that the SAS/CONNECT spawner is configured in the SERVICES file and use the following syntax to sign on:

```
options remote=<host-name>.<port-number> | <service-name>
```

The spawner service can be configured in the client's SERVICES file, but this is not a requirement. The spawner can use a port number or have the port information (service or port) defined in metadata. You can explicitly specify the port on the command line and in the SIGNON statement.

The name of the server can be specified in an OPTIONS statement:

```
OPTIONS REMOTE=node-name[.service-name | .port-number];
```

You can also specify it directly in the SIGNON statement or command:

```
%LET myNode=node-name | port-number;
SIGNON myNode;
```

node-name is based on the name of the server that you are connecting to. *node-name* must be a valid SAS name that is 1 to 8 characters in length and is either of the following:

- the short machine name of the server that you are connecting to. This name must be defined in the HOSTS file in the client operating environment or in your Domain Name Server (DNS).
- a macro variable that contains either the IP address or the name of the server that you are connecting to.

For information about configuring the SERVICES file, see "Configure the TCP/IP Services File" on page 309.

You specify *service-name* when connecting to a server that runs a spawner program that is listening on a port other than the Telnet port. If the spawner was started by using the -SERVICE spawner option, you must specify an explicit *service-name*. The value of *service-name* and the value of the -SERVICE spawner option must be identical. Alternatively, you can specify the explicit *port number* that is associated with *service-name*.

Note: If you install more than one spawner as a service on the same machine, then you must use the -NAME option to give each spawner service a unique name.

In the following example, REMHOST is the name of the node that the spawner runs on, and PORT1 is the name of the service that is defined at the client. The client service PORT1 must be assigned to the same port that the spawner is listening on.

```
%let myNode=pc.rem.us.com;
signon myNode.port1;
```

In the following example, the macro variable REMHOST is assigned to the fully qualified name of the machine that the server runs on. This server has a spawner

that was started using the following command: `cntspawn -service 5050` Therefore, the spawner is listening on port 5050. The server session that is specified in the SIGNON statement uses the node name REMHOST and the value 5050, which is the explicit port value.

```
%let remhost=pc.rem.us.com;
signon remhost.5050;
```

You can also assign a specific port number by including the port number in the definition of the macro variable. For example:

```
%let remhost=pc.rem.us.com 5050;
signon remhost;
```

Sign On

In this example, the spawner connects the client to the spawner named `spawner` that runs on the UNIX node RMTHOST. If there is no script file, the user ID and password are specified as options in the SIGNON statement. The value _PROMPT_ for PASSWORD causes SAS to prompt for a password at sign-on. The SIGNON statement makes the connection and starts a SAS session on the server.

```
signon rmthost.spawner  user=name pass=1234;
```

The table contains examples of valid spawner signons to a remote Windows host. For all of the examples, assume that the spawner was configured and started on the remote host. The CLIENT column contains valid sign-on statements that can be used to sign on to the remote Windows host machine.

Table 22.1 *Spawner Sign-on Examples*

CLIENT	Description
`%let r=pc.rem.us.com;` `signon r.7551;`	In this example, the client signs on using the host name **r** and the explicit port number 7551 that was used to start the spawner on the Windows remote host. The fully qualified domain name of the remote host is assigned to the macro variable **r**. The explicit port number (7551) is specified with the host name in the SIGNON statement. Note that the REMOTE= or CONNECTREMOTE= option is implicit in the SIGNON statement.
`%let r=pc.rem.us.com 7551;` `signon r user=abc123 pass=******;`	In this example, the client signs on by specifying the remote host name, the user ID, and the password in the SIGNON statement. The fully qualified domain name of the remote host and the explicit port number (7551) are both stored in the macro variable **r**. The user ID and password are specified in the SIGNON statement.

`%let r=pc.rem.us.com 7551;`
`options remote=r;`
`signon user=abc123 pass=_prompt_;`
	In this example, the client signs on by specifying the remote host name in the OPTIONS statement. The fully qualified domain name of the remote host and the port number 7551 are both stored in the macro variable **r**. The user ID and password are specified in the SIGNON

CLIENT	Description
	statement. The value for the PASSWORD option (_prompt_) causes the server to prompt the user for the password during the sign-on.

`%let r=pc.rem.us.com;`
`signon r._ _7551 user=abc123 pass=******;`
`libname myLib server=r._ _7551;`
	Again, the fully qualified domain name of the Windows remote host is assigned to the macro variable **r**. The host name and port number are specified in the SIGNON statement using the "computer-name.__port-number" format.
	A libref is defined on the server using the LIBNAME SERVER= statement. When a LIBNAME statement is used in a sign-on, the double underscore format must be used to specify the server ID for both the SERVER= option in the LIBNAME statement and the SERVER= option in the SIGNON statement.

`%let r=10.5.55.14  7551;` `signon r user=abc123 pass=******;`	The IP address of the UNIX remote host and its port (7551) are assigned to the macro variable **r**. The host name (**r**), user ID, and password are specified in the SIGNON statement.

`%let r=pc.rem.us.com;`
`filename rlink 'C:\Program Files\`
`  SASHome\SASFoundation\9.4\`
`  connect\saslink\tcpunix.scr';`
`signon r;`
	This example shows a scripted spawner sign-on from a Windows client machine to a remote Windows server. The fully qualified domain name of the Windows host is assigned to the macro variable **r**. The FILENAME RLINK statement is used to specify the sign-on script. The FILENAME statement assigns the default fileref, RLINK, to the script file located in **C:\Program Files\SASHome\SASFoundation\9.4\connect\saslink** on the client machine.

`filename rlink '..';` `signon mvshost._ _7551;`	This example shows a scripted spawner sign-on from a UNIX client machine to a remote Windows server. The FILENAME RLINK statement specifies the parent of the current folder as the path for the script file. The Windows host name and port number are specified in the SIGNON statement using the "computer-name.__port-number" format.

Sign On Using a Script

You can use a sign-on script to sign on to the spawner, or you can sign on to the spawner without a script. If you are signing on by using a script, you must specify the script that you want to use. The script file is executed by the SIGNON statement or command. To use one of the sample script files that are provided with SAS/CONNECT for signing on and signing off, use the FILENAME statement to assign the fileref RLINK to the appropriate script file:

```
FILENAME RLINK '!sasroot\connect\saslink\script-name.scr';
```

The script name is based on the server that you are connecting to. The sample scripts are installed in the **!sasroot\CONNECT\SASLINK** directory. The following table lists the scripts that are provided in SAS software.

Note: If the spawner is started by using the -NOSCRIPT option, then you cannot sign on by using a script. If there is no script, you do not need to assign the fileref RLINK in a FILENAME statement.

Table 22.2 *SAS/CONNECT Sign-on Scripts for TCP/IP under Windows*

Server	Script Name
TSO under z/OS	`tcptso.scr`
TSO under z/OS, SAS 9 or later	`tcptso9.scr`
z/OS (without TSO)	`tcpmvs.scr`
z/OS (using full-screen 3270 Telnet protocol)	`tcptso32.scr`
OpenVMS	`tcpvms.scr`
UNIX	`tcpunix.scr`
Windows	`tcpwin.scr`

Specify a User ID and Password

If you use a script when connecting to a spawner, script file processing passes the user ID and password to the server. By default, the script prompts for a user ID and password. If you sign on to a secured spawner without a script, you might be required to supply a user ID and password. If you do not use a script file, you can use the USERID= and PASSWORD= options in the SIGNON statement to send the user ID and password values to the server. Here is an example of using the USERID= and PASSWORD= options to connect to the spawner:

```
OPTIONS REMOTE=spawner-ID;
SIGNON USER=user-ID password=password;

SIGNON USER=user-ID | _PROMPT_ [ PASSWORD=password | _PROMPT_ ];
```

Here is an example:

```
options remote=rmthost.spawn;
signon user=slim password=_prompt_;
```

In this example, the TCP/IP access method is used by default. The spawner connects the client to the spawner named `SPAWN` that runs on the UNIX node RMTHOST. If there is no script file, the user ID and password are specified as options in the SIGNON statement. The value _PROMPT_ for PASSWORD causes

SAS to prompt for a password at sign-on. The SIGNON statement makes the connection and starts a SAS session on the server.

Specify Data Encryption Options for Client Sign-ons

Encryption is the process of transforming plaintext into a less readable form (called ciphertext) by using a mathematical process. The ciphertext is translated back to plaintext for anyone who can supply the appropriate key, which is necessary for decrypting (or unlocking) the ciphertext.

If an encryption service is available and is configured at the client, you can specify SAS options to encrypt the data that is transferred between your client session and your remote server session. The encryption options can be specified when you sign on to a server session.

In the following example, the NETENCRYPTALGORITHM option specifies the RC2 encryption algorithm is to be used for data transfers.

```
options netencryptalgorithm=rc2;
```

Here is a list of examples that show how to sign on using encryption options:

- "Connection of a SAS/CONNECT Client to a Windows Spawner on a SAS/CONNECT Server" in *Encryption in SAS*
- "Start-Up of a Windows Spawner on a Single-User SAS/CONNECT Server" in *Encryption in SAS*

For a list of SAS system options for Encryption, see "Spawner Options" on page 326.

SASCMD Connections on Windows

Sign On to the Same Multiprocessor Machine

Tasks

SASCMD signons can be established when you want to run multiple, independent SAS sessions asynchronously and in parallel on the same multiprocessor machine. Here is an example of a SASCMD sign-on:

1. Specify the server session.
2. Specify the SASCMD option to start SAS.
3. Sign on to the server session.

Specify the Server Session

You can specify the server session in an OPTIONS statement or in the SIGNON statement. Here is the syntax for using the OPTIONS statement to specify the server session:

```
OPTIONS REMOTE=session-ID;
```

Here is the syntax for using the SIGNON statement to specify the server session:

```
SIGNON session-ID;
```

session-ID must be a valid SAS name that is 1 to 8 characters in length, and is the name that you assign to the server session on the same multiprocessor machine. For details about the SIGNON statement, see “SIGNON” on page 127.

Specify the SASCMD Option to Start SAS

Use the SASCMD option to specify the SAS command and any additional options that you want to use to start SAS in the server session on the same multiprocessor machine.

The SASCMD option can be specified in an OPTIONS statement or in the SIGNON statement. Here is the syntax for specifying the SASCMD system option in an OPTIONS statement:

```
OPTIONS SASCMD="SAS-command" | "!SASCMD";
```

Here is the syntax for specifying the SASCMD option in the SIGNON statement:

```
SIGNON name SASCMD="SAS-command" | "!SASCMD";
```

For details, see “SASCMD=” on page 113 and “SIGNON” on page 127.

Sign On to the Server Session

The following example shows how to create multiple SASCMD sessions using the SIGNON statement and the SASCMD option. In the example, **SAS1** is the name of the server session and sas is the command that starts SAS on the same multiprocessor machine.

```
options comamid=tcp;
signon sas1 sascmd='sas';
```

In the following example, TCP is the access method, **SAS1** is the name of the server session, and sas is the command that starts SAS on the same multiprocessor machine.

```
options comamid=tcp;
signon sas1 sascmd='sas';
```

The NOSYNTAXCHECK System Option

You can specify the NOSYNTAXCHECK system option when signing on to a server session on the same symmetric multi-processing (SMP) computer that the client session is running on. This option is most useful when client and server sessions run on SMP hardware.

NOSYNTAXCHECK enables continuous processing of statements regardless of syntax error conditions. When SYNTAXCHECK is enabled, SAS uses additional resources to validate SAS statements while producing limited results.

For example, the first instance of a syntax error triggers syntax checking, which automatically sets the value of the OBS= system option to 0. Consequently, no observations can be created by subsequent SAS statements in the program. When executing SASCMD sign-ons or when executing debugged production programs that are unlikely to encounter errors, consider using the NOSYNTAXCHECK option.

Here is an example of a SAS invocation that runs on the same computer on which the client session runs:

```
signon smp sascmd="sas -nosyntaxcheck -noterminal";
```

Telnet Connections on Windows

IMPORTANT With the June 2023 hot fix, Telnet is deprecated and it is recommended that you use SAS/CONNECT Spawner for client sign-ons. The -CLEARTEXT option has been deprecated and is no longer available. For more information, see SAS Note 70114.

Tasks

- Ensure that the Telnet service is installed and enabled on the Windows client and server.
- Specify the server name.
- Specify a sign-on script.
- Sign on using the SIGNON statement.

  Example:

  ```
  filename rlink '!sasroot\connect\saslink\tcpwin.scr';
  options comamid=tcp remote=rmtnode;
  signon;
  ```

For a list of sign-on scripts, see sign-on scripts on page 409 .

Note: When you install Windows Server 2008, the files that make up the Telnet Server service are copied to your computer, but the Telnet service is disabled by default.

23

SAS/CONNECT Files

SAS/CONNECT Files and Directories

Directory Names

SAS/CONNECT runs in a variety of environments including single-user desktop configurations, single-server SAS Foundation installations, and multi-server, distributed environments. The type of environment you have, whether it is a planned deployment or simply a single-server SAS Foundation installation, determines what files are created automatically when SAS is installed and what files you should use when setting up SAS/CONNECT.

This document provides information about the SAS/CONNECT files that are provided with a SAS Foundation installation. For information about SAS/CONNECT files in a SAS Intelligence Platform environment, see Initial Configuration of the SAS/CONNECT Server in *SAS 9.4 Intelligence Platform: Application Server Administration Guide*.

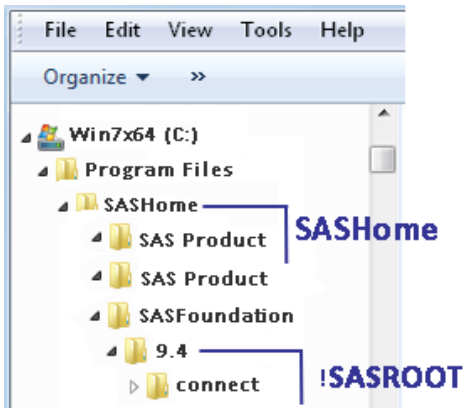
The following terms are used throughout this documentation:

SASHOME Directory
: `SASHOME` is the default installation directory for all SAS products that are installed on your system, including SAS Foundation products, SAS Intelligence Platform products, and any other SAS products.

!SASROOT Directory
: `!SASROOT` is a shorthand way of representing the subdirectory within the SASHOME directory that is the "root" of each SAS Foundation product. For example, `SASHome/SASFoundation/9.x` is the default directory for !SASROOT on UNIX, and `c:\Program Files\SASHOME\SASFoundation\9.4` is the default directory for !SASROOT on Windows.

  - UNIX: `SASHome/SASFoundation/9.x`
  - Windows: `C:\Program Files\SASHome\SASFoundation\9.x`

Figure 23.1 *Windows Example Showing SASHOME and !SASROOT Default Folders*



There is a SAS/CONNECT product folder in !SASROOT named `connect`.

SAS Foundation Installation
: an installation consisting of *SAS Foundation* and related software. Generally, this type of installation does not have a configured metadata server and does not require a customized *deployment plan*. A basic *SAS Foundation* installation runs SAS server software (along with SAS/CONNECT in this case) on a single machine in a client/server environment.

Planned Deployment
: a deployment that involves the installation and configuration of SAS servers based on an individualized plan. SAS Enterprise BI Server and SAS Data Integration Server are examples of products that are deployed using a deployment plan. In this document, "SAS Intelligence Platform" and "planned deployment" are sometimes used interchangeably.

SAS/CONNECT Files

Table 23.1 *SAS/CONNECT Files in a SAS Foundation Installation*

Name	Description	Default Location
`SASHome`	the default directory for all SAS products that are installed on your system.	location is site-specific
`!SASROOT`	represents the subdirectory within the SASHOME directory that contains the "root" of each SAS Foundation product (including SAS/CONNECT).	**Windows**: `SASHome\SASFoundation\9.x\` **UNIX**: `SASHome/SASFoundation/9.x/`
`cntspawn`	the SAS/CONNECT spawner executable used to install, start and stop the SAS/CONNECT spawner. The file is executed by specifying the CNTSPAWN command along with SAS/CONNECT spawner start-up options. This file is created when you install SAS/CONNECT. See "Spawner Options" on page 326 to see the options that you can specify with the spawner start-up file.	**Windows**: `!SASROOT\cntspawn.exe` **UNIX**: `!SASROOT/utilities/bin/cntspawn` **z/OS**: located in the SAS load module library.
`logconfig.xml` (sample on page 423)	an XML file used with the SAS Logging Facility that specifies and configures loggers and appenders for the SAS/CONNECT spawner. Once created, this file should be saved in the server's configuration directory. For a sample logconfig.xml file, see "Sample Logging Configuration File" on page 423.	**Note:** This file is not created automatically when you install SAS/CONNECT in a SAS Foundation installation. You can use this sample file and customize it according to your environment. For more information about this file, see "-LOGCONFIGLOC <*filename*>" on page 327.
`SERVICES` file (sample on page 311)	an ASCII file that provides a mapping between service names and their assigned ports. It also includes the protocol name, alias name,	**UNIX**: `/etc/services` **Windows**: `%WINDIR%\system32\drivers\etc\services` **z/OS**: `ETC.SERVICES`

Name	Description	Default Location
	and a description of the service.	
sign-on script files (samples on page 409)	sample script files provided with SAS/CONNECT software that can be used to start and stop SAS/CONNECT.	**Windows**: `!SASROOT\connect\saslink` **UNIX**: `!SASROOT/misc/connect` **z/OS**: `prefix.CTMISC`

Table 23.2 z/OS Sample Files for the SAS/CONNECT Spawner

Location	Name
`'&prefix.BAMISC(SPNCCNTL)'`	Spawner started task procedure that defines PARMS options and executes the UNIX System Services `/bin/tso` command to start SAS.
`'&prefix.BAMISC(SPNCPARM)'`	PARMS file that specifies spawner start-up options
`'&previx.BAMISC(SPNCSHEL)'`	USS Shell Script creates the REXX command to start SAS.

SAS/CONNECT Sign-on Script Files

Introduction to SAS/CONNECT Script Files

A SAS/CONNECT *script* is a SAS program that is stored in a file on the client. However, the programming statements in a script are not the usual SAS programming statements. Scripts use a specialized set of SAS statements called *script statements*.

Scripts are executed to start or to stop SAS/CONNECT sessions. Scripts that start the connection are executed by submitting the SIGNON statement, and scripts that stop the connection are executed by submitting the SIGNOFF statement. In most cases, the same script is used to sign on and sign off.

Location of Sample Scripts Included with SAS

You can start and stop SAS/CONNECT by using the supplied sample scripts, which are located in the following default directories where your SAS software is installed.

Table 23.3 *SAS/CONNECT Sample Sign-on Scripts*

Server/Operating System	Script Name	Location
TSO under z/OS	`tcptso.scr`	prefix.CTMISC
TSO under z/OS, SAS 9 or later	`tcptso9.scr`	
z/OS (without TSO)	`tcpmvs.scr`	
z/OS (using full-screen 3270 Telnet protocol)	`tcptso32.scr`	
UNIX	`tcpunix.scr`	`!SASROOT/misc/connect/`
Windows	`tcpwin.scr`	`!SASROOT\connect \saslink\`

All sample scripts start and stop SAS/CONNECT. A sign-on script prompts you for a user ID and password to sign on to a server. You must sign on to the server before you can run a manual sign-on script.

Script names are derived from the access method and the operating environment that the server session runs under. Fr example, TCPTSO.SCR identifies the TCP/IP access method and a TSO server.

See “!SASROOT Directory” on page 406 for more information about the !SASROOT directory.

Using Script Files to Sign On

There are several SAS/CONNECT language elements that you can use to specify the names and locations of sign-on scripts.

- CSCRIPT option

  specifies the SAS/CONNECT script file to use during sign-on.

  ```
  signon rhost cscript="myScript.scr";
  ```

- SASSCRIPT system option

  specifies one or more locations for SAS/CONNECT server sign-on script files and provides an alternative to the RLINK fileref that is used in the FILENAME statement for identifying the location of a script file.

```
options sasscript= "c:\my\scripts"; signon cscript=”myScript.scr”;
```

- RLINK FILENAME statement

  uses the RLINK system fileref to identify the script file to SAS. The SIGNON statement initiates the script in the RLINK file.

```
options remote=rhost;
filename rlink " \connect\saslink\tcpwinx.scr"; signon rhost;
```

- SCL functions

  describes how to use SAS Component Language functions to manage sample SAS/CONNECT files.

Purpose of a Sign-on Script

A sign-on script can be a simple, short program or a long, complex program, depending on what you want the script to do. The basic functions of all scripts are the following:

1. invoke SAS on the server (by using the SAS command).
2. set the appropriate options for the server session in the SAS command. For the server session, the script sets the “DMR”.
3. determine when the server session is ready for communications with the client session. In most cases, the script waits for messages from the server session.

Sign-on scripts might also perform the following tasks:

- issue the server sign-on command and prompt the user for a user ID and a password.
- issue informative messages to the user about whether script execution is proceeding successfully.
- combine the SIGNON and SIGNOFF functions.
- conditionally execute labeled portions of the script so that one script can accommodate multiple types of connections (for example, TCP/IP connections to both a spawner and a Telnet daemon).
- issue server commands, such as commands that set session features or define server files.
- define any response that is expected from the server.
- conditionally execute script subroutines to handle successful operations and error conditions.

Note: Scripts that sign on to the server include information that is specific to the computing installation. The scripts might need minor modifications to work with your sign-on sequence.

Passwords in a Script File

Passwords can be specified for a script file in any of these forms:

- a clear-text password that is hardcoded into the script
- a prompt for a user-supplied password as input to the script
- an encoded password that replaces a clear-text password in the script is supported only for the SAS/CONNECT Spawner.

The first and second forms offer the least security. The last form promotes security and enables you to store SAS programs that do not contain clear-text passwords.

To obtain an encoded password, specify the clear-text password in the PROC PWENCODE statement. For complete details about PROC PWENCODE, see "PWENCODE Procedure" in *Encryption in SAS* in *Base SAS Procedures Guide*.

Here is an example of code that is used to obtain an encoded password:

```
proc PWENCODE in="My2008PW";run;
{sas001}TXkyMDAzUFc=
```

The clear-text password `My2008PW` is specified in the PROC PWENCODE statement. The output is generated in the form {*key*}*encoded-password*, where sas001 is the key and TXkyMDAzUFc= is the encoded password that is generated. SAS/CONNECT uses the key to decode the encoded password to its clear-text form when the password is needed.

Note: The encoded password is case-sensitive. Use the entire generated output string, including the key.

Substitute the encoded password for the clear-text password in a script. The encoded password is the output that is generated from the PROC PWENCODE statement.

Note: Macro variables can also be used in script files to capture different user IDs and passwords. This eliminates the need for prompting the user for this information. Enclose the macro variable in double quotation marks in the script.

SAS/CONNECT Script Statements

SAS/CONNECT script statements can be used to write customized sign-on scripts or to update existing sample script files that are supplied with SAS. If the available sample scripts do not meet your requirements, you can write your own script. For script statement syntax, see "SAS/CONNECT Script Statements" on page 293.

Here is a table that provides a summary of SAS/CONNECT script statements.

Table 23.4 *Summary of SAS/CONNECT Script Statements*

Statement	Purpose
ABORT on page 293	Stops execution of a script immediately and signals an error condition.
CALL on page 294	Invokes a routine.
ECHO on page 294	Controls the display of characters that are sent from the server session while a WAITFOR statement executes.
GOTO on page 295	Redirects execution to the specified script statement.
IF on page 295	Checks conditions before the execution of labeled script statements.
INPUT on page 296	Displays a prompt to the user that requests a response for the server session.
LOG on page 297	Sends a message to the client session SAS LOG window.
NOTIFY on page 298	Sends a message in a window to the client session.
RETURN on page 298	Signals the end of a routine.
SCANFOR on page 299	Specifies a pause until conditions are met (an alias for WAITFOR).
STOP on page 299	Stops execution of a script under normal conditions.
TRACE on page 300	Displays script statements as they execute.
TYPE on page 300	Sends characters to the server session as if they were entered at a terminal.
WAITFOR on page 302	Specifies a pause until conditions are met.

Simple SAS/CONNECT Scripts for Sign On and Sign Off

Overview

When you write or modify existing SAS/CONNECT scripts, use the WAITFOR and TYPE statements to specify the sequence of prompts and responses for the server.

The simplest method for determining the sequence is to manually reproduce on the server the process that you want to capture in the WAITFOR and TYPE statements.

For each display on the server, choose a word from that display for the WAITFOR statement. Whatever information you enter to respond to a display should be specified in a TYPE statement. Be sure to note all carriage returns or other special keys.

If the server runs under z/OS and you need to use a TYPE statement that has more than 80 characters in a sign-on script, divide the TYPE statement into two or more TYPE statements. To divide the TYPE statement, insert a hyphen (-) at the division point. The z/OS server interprets the hyphen as the continuation of the TYPE statement from the previous line. For example, here is how to divide the following TYPE statement:

```
type
"sas options ('dmr comamid=tcp')"
enter;
```

change it to the following:

```
type "sas options ('dmr comamid=-" enter;
type "tcp')" enter;
```

Note: Do not insert spaces before or after the hyphen.

Syntax Rules for SAS/CONNECT Script Statements

To write a SAS/CONNECT script, you need to read about the specific information for each statement in the script. This section contains general rules that apply to some or all script statements.

- Each script line is limited to 8192 characters.
- All script statements must end with a semicolon.
- Script statements have a free format, which means that there are no spacing or indention requirements. A statement can be split across several lines, or one line can contain one or more statements. Statement keywords can be specified in uppercase, lowercase, or mixed-case characters.
- Text strings that are enclosed in quotation marks are case sensitive. For example, if your script defines a text string in a WAITFOR statement, ensure that the uppercase and lowercase characters in the text string exactly match the text string from the server.
- Any script statement can include a label specification. The label must be a valid SAS name and not exceed a maximum of eight characters. The first character must be an alphabetic character or underscore. A label must be followed immediately by a colon (:) and must be defined only one time in the script.
- Some script statements specify a time in seconds. The form of the time specification is as follows:

n SECONDS;

n can be any number; this number might include decimal fractions. For example, all of the following time specifications are valid:

- 0 SECONDS;
- 0.25 SECONDS;
- 1 SECOND;
- 3.14 SECONDS;

Note: SECOND is an alias for SECONDS.

- If a script statement specifies a quoted string, such as a server command, you can use either single or double quotation marks. To embed quotation marks in script statements, follow the same rules that you use for embedded quotation marks in SAS statements.

Debug a SAS/CONNECT Script

When writing SAS/CONNECT scripts, you can take advantage of programming techniques to simplify debugging a new or a modified script. Examples of debugging statements follow:

- The ECHO statement causes server messages to be displayed while a WAITFOR statement executes. This enables you to monitor activity on the server during the WAITFOR pause.
- The TRACE statement enables you to specify that some or all script statements be displayed as the script executes. This capability can help you isolate the source of a script problem.

Example SAS/CONNECT Script for a TCP/IP Connection to UNIX

```
/* trace on; */
/* echo  on; */
   /
*************************************************************************
/
   /* Copyright (C)
1990                                                  */
   /* by SAS Institute Inc., Cary
NC                                      */
   /
*
*/
   /* name:
tcpunix.scr                                                 */
   /
*
*/
   /* purpose: SAS/CONNECT SIGNON/SIGNOFF script for connecting to
any    */
```

```
   /*          UNIX operating environment via the TCP/IP access
method    */
   /
*
*/
   /* notes:      This script might need to be modified for
your          */
   /*             UNIX environment. The logon procedure should
mimic      */
   /*             the tasks that you execute when connecting
to           */
   /*             the same UNIX operating
environment.                    */
   /
*
*/
   /* assumes:    The command to execute SAS in your remote
(UNIX)        */
   /*             environment is "sas". If this is incorrect for
your     */
   /*             site, change the contents of the line that
contains     */
   /*             type
'sas ...                                          */
    /
*                                                                      */
   /
*
*/
   /
**************************************************************************
/

 1 log "NOTE: Script file
               'tcpunix.scr' entered.";

   if not tcp then goto notcp;
 2 if signoff then goto signoff;

   /*****************************************/
   /*  TCP/IP SIGNON                        */
   /*****************************************/

 3 waitfor 'login:', 120 seconds: noinit;

   /*****************************************/
   /*  UNIX LOGON                           */
   /* LF is required to turn the line       */
   /* around after the login name has       */
   /* been typed. (CR will not do)          */
   /*****************************************/
 4 input 'Userid?';
   type LF;
 5 waitfor 'Password', 30 seconds : nolog;
   input nodisplay 'Password?';
   type LF;
```

```
unx_log:
   /*****************************************/
   /* Common prompt characters are $,>,%,}    */
   /*****************************************/
 6 waitfor '$', '>', '%', '}',
      'Login incorrect'      : nouser,
      'Enter terminal type'  : unx_term,
      30 seconds             : timeout;


   log 'NOTE: Logged onto UNIX...
              Starting remote SAS now.';

      /*************************************/
      /* Invoke SAS on the server.       */
      /*************************************/
 type 'sas -dmr -comamid tcp -device
       -noterminal -nosyntaxcheck' LF;
 waitfor 'SESSION ESTABLISHED',
      90 seconds : nosas;

 log 'NOTE: SAS/CONNECT
      conversation established.';
   stop;

   /*****************************************/
   /*  TCP/IP SIGNOFF                       */
   /*****************************************/
10 signoff:
waitfor '$', '>', '%', '}',
      30 seconds;

   type    'logout' LF;
   log 'NOTE: SAS/CONNECT conversation
        terminated.';
   stop;

   /*****************************************/
   /*  SUBROUTINES                          */
   /*****************************************/
unx_term:

      /*************************************/
      /* Some UNIX systems want the          */
      /* terminal-type. Indicate a basic     */
      /* tele-type.                          */
      /*************************************/
   type 'tty' LF;
   goto unx_log;

   /*****************************************/
   /*  ERROR ROUTINES                       */
   /*****************************************/
11 timeout:
   log 'ERROR: Timeout waiting for remote
```

```
      session response.';
   abort;

nouser:
   log 'ERROR: Unrecognized userid or
               password.';
   abort;

notcp:
   log 'ERROR: Incorrect communications
               access method.';
   log 'NOTE: You must set "OPTIONS
              COMAMID=TCP;" before using
              this script file.';
   abort;

noinit:
   log 'ERROR: Did not understand remote
               session banner.';

nolog:
   log 'ERROR: Did not receive userid or
               password prompt.';
   abort;

nosas:
   log 'ERROR: Did not get SAS software
               startup messages.';
   abort;
```

1 The LOG statement sends the message that is enclosed in quotation marks to the log file or the log window of the client session. Although it is not necessary to include LOG statements in your script file, the LOG statements keep the user informed about the progress of the connection.

2 The IF/THEN statement detects whether the script was called by the SIGNON command or statement or the SIGNOFF command or statement. When you are signing off, the IF/THEN statement directs script processing to the statement labeled SIGNOFF. See step 10.

3 The WAITFOR statement waits for the server's logon prompt and specifies that if that prompt is not received within 120 seconds, the script processing should branch to the statement labeled NOINIT.

4 The INPUT statement displays a window with the text `Userid?` to allow the user to enter a server logon user ID. The TYPE statement sends a line feed to the server to enter the user ID to the server.

5 The WAITFOR statement waits for the server's password prompt and branches to the NOLOG label if it is not received within 30 seconds. The INPUT statement that follows the WAITFOR statement displays a window for the user to enter a password. The NODISPLAY option is used so that the password is not displayed on the screen as it is entered.

6 The WAITFOR statement waits for one of several common UNIX prompts and branches to various error handles if a prompt is not seen. Verify that the WAITFOR statement in the script looks for the correct prompt for your site.

7 This TYPE statement invokes SAS on the server. The -DMR option is necessary to invoke a special processing mode for SAS/CONNECT. The -COMAMID option

specifies the access method that is used to make the connection. The -NOTERMINAL system option suppresses prompts from the server session. The -NOSYNTAXCHECK option prevents the remote session from going into syntax checking mode when a syntax error is encountered.

8 The phrase `SESSION ESTABLISHED` is displayed when a SAS session is started on the server by using the options -DMR and -COMAMID TCP. The WAITFOR statement looks for the words `SESSION ESTABLISHED` to be issued by the server session to know that the connection has been established. If the `SESSION ESTABLISHED` response is received within 90 seconds, processing continues with the next LOG statement. If the `SESSION ESTABLISHED` response does not occur within 90 seconds, the script assumes that the server session has not started and processing branches to the statement labeled NOSAS.

9 When the connection has been successfully established, you must stop the rest of the script from processing. Without this STOP statement, processing of the remaining statements in the script continues.

10 This section of code is executed when the script is invoked to end the connection. The second IF statement (see step 2) sends processing to this section of the script when the script is invoked by a SIGNOFF command or statement. Note that this section waits for a server prompt before entering `LOGOUT` in order to log off from the server. The script then issues a LOG statement to notify the user that the connection is terminated and stops script processing.

11 These statements are processed only if the prompts expected in the previous steps are not received. This section of the script issues messages to the local SAS log and abnormally ends (from the ABORT statement) the processing of the script as well as the sign-on.

PART 5

Logging and Debugging

24

Administering Logging for SAS/CONNECT

SAS Logging Facility

About the SAS Logging Facility

The SAS/CONNECT server and the SAS/CONNECT spawner use the SAS logging facility as the standard debugging tool in a SAS Foundation environment and in a SAS Intelligence Platform deployment. To make the logging facility functional, you must define a logging configuration, which configures appenders and loggers. You can define the configuration by setting up an XML file or by using SAS language elements.

In a planned deployment, the SAS Deployment Wizard creates `logconfig.xml` files for the SAS/CONNECT server and the SAS/CONNECT spawner. You can modify these configuration files as needed to adjust your logging configuration. See About

Server Logging for more information about administering logging in a SAS Intelligence Platform environment.

If you have a SAS Foundation installation, you can create your own logging configuration file using the example shown in Example Code 24.1 on page 423.

The SAS Logging Facility

1 Define your logging configuration.

   To use the SAS logging facility, you must set up your logging environment by defining a logging configuration. The logging configuration can be in the form of an XML file or a set of SAS program statements that configure how log events are processed. The logging configuration enables you to configure appenders and to specify which categories and levels of log events are written to each appender.

2 Enable the logging facility.

   To enable logging in a SAS Foundation environment, specify the -LOGCONFIGLOC system option in the SAS invocation and specify the name and location of the logging configuration file as the value for the -LOGCONFIGLOC system option:

   Here is the syntax for the -LOGCONFIGLOC option:

   -LOGCONFIGLOC <*file-specification*>

   Here is an example showing the -LOGCONFIGLOC option specified on the SAS start-up command:

```
sas -logconfigloc winlog.xml
```

   Note: In a planned deployment, the SAS/CONNECT spawner automatically enables logging in the **`ConnectSpawner.sh script`** file, and the SAS/CONNECT server enables logging in the **`sasv9.cfg`** file.

   In the example, the -LOGCONFIGLOC option is used to specify the location of the logging configuration file named **`winlog.xml`**, which is used to initialize the SAS logging facility. The file specification that defines the location of the logging configuration file must be a valid filename or a path and filename for your operating environment.

How to Change the Logging Level of the SAS/CONNECT Spawner

By default, the SAS/CONNECT spawner does not write much information to its log file. Therefore, to debug SIGNON problems, you might need to change the logging level so that it reports more detailed information. You can change the spawner's logging level either dynamically using spawner start-up options or you can do it permanently by updating the logging configuration file. To do this dynamically,

specify either the -DEBUG or the -TRACE (–VERBOSE) option on the spawner start-up command as shown here:

```
cntspawn -verbose -logfile /local/u/sasusr/mytest/spawner.log  -service unxspawn
```

To change the logging level permanently, update the logging configuration file. Open the spawner's logging configuration file and change the value of the logger's level tag:

```
<logger name="App">
<level value="Trace">
</logger>
```

For more information, see "-DEBUG" on page 327 and "-TRACE|VERBOSE" on page 331 spawner options.

CAUTION

Excessive logging can degrade performance. Therefore, you should use the TRACE and DEBUG logging levels cautiously.

Sample Logging Configuration File

Here is a typical configuration file that defines the spawner logging components:

Note: In a planned deployment, the SAS Deployment Wizard automatically creates an initial logging configuration file, named **logconfig.xml**, for each of your servers. If you have a Foundation-only deployment or your deployment is not a planned deployment in which the SAS Deployment Wizard was used, you can copy the sample logconfig.xml file and change it to meet your needs.

Example Code 24.1 *Sample Logging Configuration File*

```
<?xml version="1.0" encoding="UTF-8"?>
<log4sas:configuration xmlns:log4sas="http://www.sas.com/rnd/Log4SAS/"
debug="true">
   <appender name="LOG" class="FileAppender">1
      <param name="File" value="c:\v9\spawner.log" />2
      <layout>
         <param name="ConversionPattern" value="%d %-5p [%t] %c
(%F:%L) - %m" />
      </layout>
      <param name="threshold" value="all" />
   </appender>
<root>3
   <appender-ref ref="LOG" />
   <level value="info" />
</root>
</log4sas:configuration>
```

1 CLASS="FileAppender" indicates that the log events are written to the file path c:\v9\spawner.log.

2 The ConversionPattern parameter specifies a pattern layout that formats log messages. It identifies the type of data, the order of the data, and the format of

the data that is generated in a log event and is delivered as output. In this example, the date and time, the log level, the thread ID, and the logger constitute the log event.

3 The root logger controls the entire SAS log event and is at the highest level in the logger hierarchy. If any other loggers are included in the logging configuration file, they are located beneath the ROOT logger in the hierarchy. All other loggers inherit the specified appender and threshold value of the root logger.

Triggers for Log Events

Log events are triggered for various activity in SAS/CONNECT under these circumstances:

- server sign-on via the SIGNON statement and the SAS/CONNECT spawner invocation
- the beginning of the RSUBMIT statement and the occurrence of the ENDRSUBMIT statement
- server sign-off via the SIGNOFF statement and the SAS/CONNECT spawner termination

Example of a Log Event

The data and the format of the log event are defined in the conversion pattern that is specified in the configuration file.

Here is an example of a log event:

```
  2008-06-25-10:24:22,234; WARN; 3; Appender.File; (yn14.sas.c:149);
Numeric maximum was larger than 8, am setting to 8.
```

SAS Console Log

Definition

The SAS console log is a file that is created when the regular SAS log is not active. The SAS console log is used for recording information, such as warnings and error messages that occur before the regular SAS log as been initialized. Therefore, the SAS console log might contain error messages that were sent because SAS failed to start on the server.

SAS Console Log Messages for Windows

In Windows, any error message that SAS issues before the regular SAS log is initialized is written to a file that is located in the user's Application Data Directory. The name of the file is written as a record to the Windows Application Event Log.

You can use the Windows Event Viewer to see the application events on the computer where the server session was being executed. A warning event is logged for each initialization failure for a single server session. For multiple events, the user ID and the time of the event are included in the warning event.

For more information about the failing event, you can select the warning event from the viewer window. Another window is displayed that contains detailed event information, including the name of the file that contains the SAS console log. To open the Windows Application Event Viewer, submit `eventvwr` from the **Run** dialog box.

SAS Console Log Messages for UNIX

On UNIX, the SAS console log is written to the standard output location for the SAS process. The location for the standard output varies according to the sign-on method that was used.

Table 24.1 *Location of the SAS Console Log on UNIX Based on the Sign-on Method*

Sign-on Method	SAS Console Log Location
SASCMD=	Standard output is piped to the SAS session that issued the sign-on statement. The standard output messages are written to the SAS log on the SAS session. Each message contains a prefix that identifies the server session (the server ID) that was being created.
Spawner	The standard output location for the SAS session that is started via the spawner is piped to the standard output location of the spawner. The command that is used to start the spawner should ensure that standard output is redirected to a specific location. Here is an example of redirecting standard output to a log file in UNIX: `cntspawn -nocleartext > spawner.log` SAS console log messages will be directed to the standard output location. For details about the UNIX spawner, see “Spawner Connections on UNIX” on page 345.
Telnet daemon	The standard output location for the SAS session is the script processor in the SAS session that issued the SIGNON command. If the script processor does not receive a SESSION STARTED message from the server session, a sign-on failure is assumed. However, error

Sign-on Method	SAS Console Log Location
	messages that are directed to the SAS console log on the server session might not be displayed. To display error messages in the server session, include the `echo on` statement in the sign-on script.

SAS Console Log Messages for z/OS

The SAS console log receives messages that are intended for the SAS log but must be written before the SAS log is opened. Thus, it is normally empty because no such messages are written in a successful session.

The main use of the SAS console log is for error diagnostics when SAS terminates without ever opening the SAS log. For example, if an invalid command-line option prevents SAS from completing initialization, error messages would be sent to the SAS console log.

The SAS console log is written to the SASCLOG `ddname`. In a local interactive session, SASCLOG is normally allocated to the terminal. A remote session has no terminal, so SASCLOG has to be allocated differently.

Table 24.2 *Location of the SAS Console Log on z/OS Based on Sign-on Method*

Sign-on Method	SAS Console Log Location
SASCMD=	Standard output is piped to the SAS session that issued the sign-on statement. The standard output messages are written to the SAS log on the SAS session. Each message contains a prefix that identifies the server session (the server ID) that was being created.
Spawner	SAS console log output is written to the SAS/CONNECT spawner log by default. For the output to be written to a file that is owned by the user instead of by the spawner, edit the shell script that is specified on the spawner's -SASCMD option and add the -CLOG option to the SAS command. For example, the following command can be added to the shell script file: `cmd="$cmd -clog connect.sasclog"` **Note:** the data set name (or UFS name) must be one that will be writable for all users. Do not use a fully qualified name.
Telnet daemon	The SASCLOG `ddname` is directed to the script processor in the SAS session that issued the SIGNON command. If the script processor does not receive a SESSION STARTED message from the server session, a sign-on failure is assumed. However, error messages that are directed to the SAS console log in the server session might not be displayed. To

display error messages in the server session, include the ECHO ON statement in the sign-on script.

25

TCP/IP Troubleshooting

UNIX: TCP/IP Access Method

SAS/CONNECT Error Messages under UNIX

For TCP/IP, if SAS/CONNECT is unable to connect to the TCP/IP port, the following system message appears:

```
connection refused
```

The connection might fail at sign-on for the following reasons:

- The remote side is not listening.
- The maximum number of connections has been reached.

z/OS: TCP/IP Access Method

SAS/CONNECT Error Messages under z/OS

For TCP/IP, if SAS/CONNECT is unable to connect to the TCP/IP port, the following system message appears:

```
connection refused
```

The connection might fail at sign-on for the following reasons:

- The remote side is not listening.
- The packet sequence is out of order, which can indicate that the routers are not working properly.
- The maximum number of connections has been reached.
- There is a flow problem, which indicates that too many packets are being sent to the remote side at the same time.

Under z/OS, use the NETSTAT utility to show active sockets and to show who is waiting for a socket.

Windows: TCP/IP Access Method

SAS/CONNECT Error Messages under Windows

For TCP/IP, if SAS/CONNECT is unable to connect to the TCP/IP port, the following system message appears:

```
connection refused
```

The connection might fail at sign-on for the following reasons:

- The remote side is not listening.
- The maximum number of connections has been reached.

26

Sign-On Troubleshooting

Troubleshooting Sign-On Problems

Host-Not-Active Message

While signing on to a server session, you receive the following message:

```
ERROR:  Did not get Host prompt.
        Host not active.
```

If you are signing on to computer via a TCP/IP connection, one of the following actions might overcome the problem:

- Look at the script that you used for signing on. Ensure that the character string in the WAITFOR statement that tests for the server session system prompt exactly matches the character string that normally appears in the server session. The WAITFOR statement is case sensitive.
- Look at the value of the REMOTE= option in the client session to be sure it specifies the correct IP address.
- If you do not find any errors after checking the two preceding items, modify the script file by adding a TRACE ON statement and an ECHO ON statement at the beginning of the script file. These statements send a copy of the remote screen to the Log window or to a file in the client session. You can examine the SAS log on the client session to see what is displayed by the server session when the WAITFOR statement executes.

Absence of SAS Software Start-Up Messages

While signing on to a server session, you receive the following message:

```
ERROR:  Did not get SAS software startup messages
```

This message occurs if the command to invoke the server session is not correct in the script file that is being used for signing on. Look at your script file and make sure that the TYPE statement that invokes SAS in the server session uses the correct SAS command for your site. At some sites, the command to invoke SAS is not the default command name SAS.

For more information about recovery from this error, see “SAS/CONNECT Server Session Initialization Errors” on page 432.

Requested-Link-Not-Found Message

When you sign on to a server session from a client session that runs under z/OS, you receive the following message:

```
ERROR:  XMS Communication Failure:
          requested-link XVT not found.
```

This error occurs if XMS has not been configured correctly. For details about XMS configuration, see Configuration Guide for SAS® 9.4 Foundation for z /OS.

For more information about recovery from this error, see “SAS/CONNECT Server Session Initialization Errors” on page 432.

SAS/CONNECT Server Session Initialization Errors

The method that you used to sign on to a server session correctly executed the SAS command to start the server session. However, errors prevent SAS from initializing. Possible explanations for initialization failure include the following:

- An invalid option name or value might have been specified in the SAS command.
- The user might not be authorized by the computer that the server session runs on to execute the SAS program modules or to access the Sashelp, Sasuser, or SASWork libraries
- The sign-on command might try to execute an autoexec file that does not exist.

In order to recover from the initialization failure, you need to view the content of the SAS console log. The location of the SAS console log varies according to the operating environment that the server session runs under.

Cannot-Start-Remote-Process Message

While signing on to a server session, you receive the following message:

```
Error: Cannot start remote process
```

This message might occur if a SAS Viya spawner is used with the `-nolocallaunch` option and the client does not support it.

27

Compute Services Troubleshooting

Problems and Solutions When Using the RSUBMIT Statement

Invalid Option

The first time that you remote submit a PROC statement, you receive the following message:

```
ERROR 2-12:  Invalid option.
```

The remote AUTOEXEC.SAS file contains an OPTIONS statement that has not been closed by a semicolon (;). To recover from the problem, add the semicolon (;) to the OPTIONS statement in the remote AUTOEXEC.SAS file.

Dialog Box Appears despite NOTERMINAL Option Setting

Despite your setting the NOTERMINAL option to suppress the display of a dialog box in the server session, a dialog box appears when you use the RSUBMIT statement and the WAIT= option.

To prevent the appearance of a dialog box, specify the SAS system option NOFILEPROMPT in the server session.

Remotely Submitted Statements Following a Syntax Error Are Not Processed

When a SAS/CONNECT session is started and the NOTERMINAL option is set, the internal option SYNTAXCHECK is automatically set. If you remote-submit a statement that follows a syntax error, the statement is parsed but is not processed.

An example of the problem and recovery follows:

```
data a;
   do i=1 to 10;
      outpt;
   end;
run;
data b;
   x=1;
run;
```

Data set A is not created because of the syntax error that is caused by the misspelling of the word "OUTPUT." Data set B is not created because SAS is in syntax check mode from the previous syntax error. Only the DATA step will be parsed.

To prevent this problem, add the NOSYNTAXCHECK option to the server session SAS invocation options in the script file.

Square Bracket Keys Not Supported

You cannot remotely submit code that uses square brackets because the local computer's keyboard does not support these characters.

The less than (<) and greater than (>) symbols can be used in place of square brackets. Use < for the left square bracket ([), and use > for the right square bracket (]).

No Terminal Connected to SAS Session

After remotely submitting code that generates a full screen, you receive the following message:

```
ERROR:  No terminal connected to the SAS session.
```

SAS/CONNECT does not support remote submission of a window. You might be able to issue a LIBNAME statement, and use the windowing product in the client session while accessing the remote data.

Piping Problems

MP CONNECT pipeline processing can fail if the procedure that reads from the pipe (output pipe) finishes processing before the procedure that writes to the pipe (input pipe). The premature termination of the pipe causes the procedure that writes to the pipe to fail.

The error message varies according to the specific procedure that is being performed.

To prevent a pipe from terminating prematurely, assign sufficient processing time for each procedure by specifying the TIMEOUT= option in the LIBNAME statement. Furthermore, if the OBS= option in the appropriate procedure is used to limit the amount of data that is read from a large data set that is being written, processing will finish for the read procedure before the write procedure. To prevent the pipe from terminating, assign a longer time-out for the read procedure than the write procedure. For a program example, see “Example 7: Prevent Pipes from Closing Prematurely” on page 65.

Request for Setup of Link for Communication Subsystem Partner Fails

When you attempt to connect to a server session, you receive the following error message:

```
ERROR: A communication subsystem partner link setup request failure has occurred.
```

A possible explanation for the failure is that the spawner has not been started on the remote computer that you are trying to sign on to. For details about starting a spawner, see Chapter 19, “The SAS/CONNECT Spawner,” on page 319.

Another possibility is that you have used the same task name for multiple jobs that you have submitted for asynchronous processing on the same host or on a different host across the network. Task names must be unique.

28

Data Transfer Services Troubleshooting

Troubleshooting the UPLOAD and DOWNLOAD Procedures

Symbol Is Not Recognized

During a PROC DOWNLOAD or a PROC UPLOAD step, you receive the following error message:

```
ERROR 200-322: The symbol is not recognized.
```

This problem occurs if the file on the server that is being referenced by the INFILE= or the OUTFILE= option begins with a special character and is specified as FILEREF(filename). For example:

```
PROC UPLOAD INFILE=pcflref
    OUTFILE=hstflref($filname);
 run;
```

To avoid the problem, enclose the filename in single quotation marks, as shown in the following example:

```
PROC UPLOAD INFILE=pcflref
    OUTFILE=hstflref('$filname');
```

```
run;
```

Variable-Block Binary File LRECL Value Exceeds 256

You transfer a variable-block binary file that has a record length (LRECL) that is greater than 256 bytes, and SAS/CONNECT segments the file into multiple 256-byte records. For example, downloading a binary file that has an LRECL of 1024 results in four 256- byte records.

The data is not lost when the file is segmented by SAS/CONNECT. Using the LRECL option in the FILENAME statement that is processed at the client or the server does not prevent the problem. To solve the problem, follow these steps:

1. Define the z/OS FILENAME statement by using the RECFM=U parameter.

   ```
   FILENAME VFILE 'VARIABLE.BLOCK.FILE' RECFM=U;
   ```

2. Use the DOWNLOAD procedure with the BINARY option to transfer the file. Information about the transfer that is displayed in the local Log windows shows how many bytes were transferred. For example:

   ```
   NOTE:  1231 bytes were transferred at
   1231 bytes/second.
   ```

3. At the client, use the RECFM= and the LRECL= options in the INFILE statement that is used to read in the transferred file, where RECFM= is set to S370VB and LRECL= is set to the number of bytes that are transferred.

Note: In SAS 9.4, the default value for LRECL is 32767. If you are using fixed length records (RECFM=F), the default value for LRECL is 256.

Fixed-Block Binary File LRECL Value Exceeds 256

You transfer a fixed-block binary file that has a record length (LRECL) that is greater than 256 bytes, and SAS/CONNECT segments the file into multiple 256–byte records. For example, downloading a binary file that has an LRECL of 1024 results in four 256- byte records.

The data is not lost when the file is segmented by SAS/CONNECT. Using the LRECL= option in the FILENAME statement at the client or the server does not prevent the problem. To solve the problem, follow these steps:

1. Use the DOWNLOAD procedure with the BINARY option to transfer the file.
2. The INFILE statement that is used to read the transferred file must contain the options RECFM=F and LRECL=*xxxx*, where *xxxx* is equal to the LRECL parameter at the server.

Note: In SAS 9.4, the default value for LRECL is 32767. If you are using fixed length records (RECFM=F), the default value for LRECL is 256.

Note: The RECFM= and LRECL= options in the FILENAME statement are supported only under z/OS operating environments. For details, see the "FILENAME Statement: z/OS" in *SAS Companion for z/OS*.

EBCDIC CC-Control Is Not Downloaded

When you use PROC DOWNLOAD on a print file, the EBCDIC carriage-control character `'F1'x` is not downloaded.

To avoid the problem, change the SAS system option FILECC to NOFILECC.

Note: The FILECC system option is supported only under z/OS operating environments. For details, see "FILECC System Option: z/OS" in *SAS Companion for z/OS*.

The NOFILECC option indicates that the data in column 1 of a printer file should be treated as data and not carriage control. Releases of SAS later than SAS 6 use FILECC as the default setting, which you must change to NOFILECC in order to successfully download `'F1'x`. In addition, the DCB characteristics of the print file must include a value for RECFM= of FBA or VBA.

PART 6

Appendix

Appendix 1

Cross-Architecture Issues

Translation of SAS Data between Computers That Represent Data Differently

Overview of Data Translation between Computers

SAS/CONNECT clients and servers can access SAS data and programs from each other, despite differences in how data is represented on computers that the client and server SAS sessions run on. For example, a SAS/CONNECT client that runs on a PC can download a SAS data set from a mainframe for processing in the client session.

Numeric data (floating-point representation) and character data are dynamically translated in each client/server transfer. This process bypasses the explicit creation of an intermediate transport file without the user's knowledge of the underlying translation activities.

Remote Library Services

Remote Library Services (RLS) performs dynamic data translation. SAS/CONNECT use RLS to access SAS files in remote SAS libraries. SAS/CONNECT clients access remote files by using the LIBNAME statement.

Note: You can also use the CONNECT TO statement in PROC SQL to access remote files.

If the server data is accessed and processed to produce a single result at the client, only one translation occurs: from the representation of the server computer to the representation of the client computer.

If the server data is processed on the client and the results are updated on the server, two translations occur.

- When the data is accessed from the server, it is translated from the representation of the server computer to the representation of the client computer.
- When the data is updated (and stored) on the server, it is translated from the representation of the client computer back to the representation of the server computer.

Depending on the characteristics of the data, translation can cause a loss of some degree of numeric precision and magnitude.

The LIBNAME statement can be used to identify the server library to be accessed. Various SAS statements can be used to process the data, specifying the location of the server data and methods of data processing. These examples show that data is read (and translated) from the server and processed, and that the results are copied to a client location.

```
libname  serv-libref 'server-library'
server=server-ID;
libname client-libref 'client-library';
proc copy in=serv-libref
out=client-libref;
```

Note: Using RLS in a SAS/CONNECT session is not the most efficient method to move large quantities of server data. RLS is used here to illustrate the possibility for the loss of precision across computers that represent numeric data differently.

For details about how to access a remote file system, see Chapter 4, “Using Remote Library Services (RLS),” on page 71.

Data Transfer Services

Overview

Data Transfer Services (DTS) performs dynamic data translation. SAS/CONNECT uses DTS to upload and download complete or partial SAS files in a client/server environment.

For an upload, the client sends data to the server for processing. For a download, the client requests the transfer of data from the server to the client for processing.

For more information, see Chapter 5, “Using Data Transfer Services,” on page 87.

The translation process for transferring data varies according to the SAS release.

Translation of SAS 8 and Later Releases

In SAS 8 and later releases, translation occurs only once for each data transfer between a client and a server that run on computers whose architectures are different from each other. SAS/CONNECT dynamically translates incompatible file formats for each file upload or file download transaction, bypassing the explicit creation of a transport file.

LIBNAME statements are used to identify the server library to be accessed and the client library that the server data is written to. PROC DOWNLOAD reads the data from the server and translates and copies it to a specified client location.

```
libname client-libref ' client-library';
rsubmit;
   libname serv-libref ' server-library';
   proc download
data=server-libref.data-set
      out=client-libref.data-set;
endrsubmit;
```

SAS 6 Translation

In SAS 6, translation occurs twice for each data transfer between a client and a server that run on computers whose architectures are different from each other.

1. The data is translated from the source computer's native format to transport format.
2. The data that is represented in transport format is translated to the target computer's native format.

LIBNAME statements are used to identify the server library to be accessed and the client library that the server data is written to. PROC DOWNLOAD translates the data from the server into transport format, which is next translated to the client computer format when copied to a specified client location.

```
libname client-libref ' client-library';
rsubmit;
   libname serv-libref ' server-library';
   proc download
data=server-libref.data-set
      out=client-libref.data-set;
endrsubmit;
```

Translation of Floating-Point Numbers between Computers

Loss of Numeric Precision and Magnitude

If you move SAS data between a client and a server session that run on computers that have different architectures, numeric precision or magnitude can be lost. Precision can be lost when the data value in the source representation contains more significant digits than the target representation can store. A loss of magnitude results when data values exceed the range of values that an operating environment can store.

For complete details about how SAS stores numeric values, see *SAS Language Reference: Concepts*.

Avoiding Loss of Precision

To avoid loss of precision, do not store numeric values in short variables. Instead, store numeric values using longer numeric variables (up to 8 bytes) according to the number of significant digits that the target representation can store.

Significance of Loss of Magnitude

When you lose magnitude, SAS produces the following warning:

```
WARNING:  The magnitude of at least one numeric value
was decreased to the maximum the target representation allows,
due to representation conversion.
```

A loss of magnitude is unlikely in many applications. If you have data with extremely large values or extremely small fractions, then you might experience a loss of magnitude during cross-architecture access. When you lose magnitude, SAS changes the values that are out of range to the maximum or minimum value that the operating environment can represent.

Table A13.1 *Approximate Value Ranges by Operating Environment*

Operating Environment	Minimum Value	Maximum Value
UNIX	2.3E-308	1.8E+308
Windows	2.3E-308	1.8E+308
z/OS	5.4E-79	7.2E+75

Example

You create a data set under UNIX that contains the value `8.93323E+105`. If you copy the file to a z/OS operating environment, magnitude is lost and the value changes to `7.23701E+75`, which is the maximum value that z/OS can represent.

Encoding Compatibility between SAS/CONNECT Client and Server Sessions

To successfully use SAS/CONNECT programming services, the encodings of the client and server sessions must be compatible. Transport data has an encoding family dependency, so the encodings of the client and server session should be compatible in order to ensure the data will not be corrupted during the transmission. Compatible encodings share a common character set. For example, client and server sessions that each use the UTF-8 encoding should be compatible with each other.

Beginning with version 9.4, SAS/CONNECT supports connections between the client and the server in which one session is using UTF-8 and the other is using non-UTF-8. However, if one session's encoding is not compatible with the other session's encoding, then SAS issues a NOTE stating that data might not have been transmitted correctly. In the case where one session is using UTF-8 and the other session has an unknown, or unsupported encoding, an error is issued and the connection is not made. Similarly, Japanese characters and Chinese characters are both defined in the Unicode standard, but because their code points are different in

Unicode, the Japanese SAS session is not compatible with the Simplified Chinese SAS session.

In some cases, a client session can connect to a server session even though each session runs in a different locale and neither uses the UTF-8 encoding. If each session's encoding contains all the characters of each locale's native language, then the sessions are compatible and a connection occurs. For example, a Windows client session that uses the Wlatin1 encoding that is associated with the Spanish Mexico locale is compatible with a UNIX server session that uses Latin1 encoding that is associated with the Italian Italy locale. All the characters used in the Italian and Spanish languages are present in both the Wlatin1 and the Latin1 encoding.

However, SAS/CONNECT programming services might not successfully run in incompatible client and server sessions. For example, a client session that uses the Wlatin2 encoding that is associated with the Czech Republic locale is incompatible with the server session that uses the open_ed-1141 z/OS encoding that is associated with the German Germany locale. The Wlatin2 encoding and the open_ed-1141 encodings are not compatible because many German characters are not present in the Wlatin2 encoding and many Czech characters are not present in the open-ed-1141 encoding. The operation might not be successful and a note such as the following might be sent to the log:

```
Note: The client session encoding Wlatin2  is not compatible with the
server session encoding open-ed-1141.
Data may not be transmitted correctly.
```

For information about locales and encodings, see the *SAS National Language Support (NLS): Reference Guide*.

Appendix 2

SAS/CONNECT Cross-Version Issues

Factors Affecting Access to SAS Files

SAS files (data and applications) that were created by using SAS releases later than SAS 6 are interchangeable in a SAS/CONNECT client/server environment because their file formats are identical.

However, because the SAS file formats of the newer SAS releases (after SAS 6) are dramatically different from older SAS releases (SAS 6 and earlier), the ability to access older SAS files from newer SAS releases (or newer SAS files from older SAS releases) in a SAS/CONNECT client/server environment is limited. Access is determined by the following factors:

- SAS version
- SAS member type
  - Data set
  - Catalog
  - View
- SAS/CONNECT service
  - Remote Library Services (RLS)

    CAUTION
    RLS in SAS/CONNECT 9 and later is not backward compatible with SAS 6 files. SAS/CONNECT 9 clients cannot use RLS with SAS 6 SAS/CONNECT servers. SAS 6 SAS/CONNECT clients cannot use RLS with SAS/CONNECT 9 servers.

  - Compute Services
  - File Transfer Services

For SAS release information that relates to single-user SAS mode, see the *SAS Language Reference: Concepts*.

Features Exclusive to SAS Releases after SAS 6

New Features Incompatible with SAS 6

These new features in SAS cannot be modified to make SAS files compatible with SAS 6:

- generation data sets
- integrity constraints

Any attempt to access SAS files that contain these features will fail. For complete details about new features, see *SAS Language Reference: Concepts*.

SAS File Format Features

The file format features of newer SAS releases and SAS 6 are incompatible. Here are the file format features of the newer releases:

- long data set labels
- long variable labels
- long variable names

However, in order to maintain the ability to transfer data sets between the newer and older SAS releases, SAS/CONNECT applies truncation rules to data set attributes. Truncation enables you to take advantage of the features of the newer SAS releases while continuing to access SAS 6 files in a mixed-version environment.

File Transfer Services: Truncating Long Names and Labels

The newer SAS releases support longer names and labels than the maximum length supported in SAS 6. The longer names and labels are stored in SAS 8 (or later) data sets, which make those data sets incompatible with SAS 6 data sets. SAS/CONNECT implements a set of truncation rules to convert data sets that contain long names and labels into SAS 6 data sets.

The UPLOAD or DOWNLOAD procedures apply the truncation rules when performing these types of transfers of SAS files

- from a SAS 8 (or later) SAS session to a SAS 6 SAS session
- between two sessions (each running SAS 8 or later) to produce a SAS 6 data set.

Note: To produce a SAS 6 data set explicitly, specify VALIDVARNAME=V6 in the SAS session that the data set is created in. A setting of VALIDVARNAME=V6 overrides any other engine specification in the SAS session, causing truncation to be applied to long names.

SAS/CONNECT applies the following truncation rules to data sets that have long data set labels, long variable labels, or long variable names. In each case, the length is truncated to the maximum length that is supported in SAS 6.

Table A14.1 SAS 6 Truncation Lengths

Label or Name	Truncation Length (in characters)
Data set label	40
Variable label	40
Variable name	8

Note: If the variable label field is empty, the long variable name is copied to the label field.

The truncation algorithm that is used to produce the 8-character variable name also resolves conflicting variable names. Here are some additional truncation rules:

Table A14.2 Truncation Rules to Resolve Conflicting Variable Names

Truncation Rule	Example
The first name that has more than eight characters is truncated to eight characters.	STOCKNUMBER53 is truncated to STOCKNUM.
The next name that has more than eight characters is truncated to eight characters. If it conflicts with an existing variable name, it is truncated to seven characters, and a suffix of 2 is added.	STOCKNUMBER54 is truncated to STOCKNU2.
The suffix is increased by one for each truncated name that results in a conflict. If the suffix reaches 9, the next conflicting variable name is truncated to 6 characters, and a suffix of 10 is added.	STOCKNUMBER63 is truncated to STOCKN10.

RLS: Access SAS Files in a Mixed Cross-Version Library

Separate Older SAS Files from Newer SAS Files

Whenever possible, keep older SAS files (SAS 6) and newer SAS files (created using SAS releases after SAS 6) in separate physical locations. Segregation of release-specific files avoids confusion about what files can be accessed when using RLS.

Specify an Engine to Locate Release-Specific Files in a Mixed Library

Your ability to access a specific SAS file in a library depends on the engine that is associated with that library. You can explicitly specify the engine in the LIBNAME statement, or you can allow SAS to select the appropriate engine according to the version of SAS being used and the format of the SAS files in the directory. If the library is homogenous (for example, all data files are SAS 9 files), the V9 engine is used, by default.

Note: The V9 and V8 engines provide identical functionality.

However, if a physical library contains a mixture of SAS 6 files and SAS 8 files, a SAS session that runs a newer release of SAS can use the V6 engine to access only the SAS 6 files in that library.

CAUTION
A SAS 9 session cannot access SAS 6 files in a mixed library.

If a library contains newer and older SAS files and the V9 or V8 engine is specified, only the SAS 9 or SAS 8 files can be accessed. The SAS 6 files are not recognized in the SAS 9 or SAS 8 session.

However, if the V6 engine is specified, the SAS 6 files can be accessed. The SAS 9 or SAS 8 files are not recognized.

In the following example, the libref V8LIB accesses only SAS 9 or SAS 8 files.

```
libname v8lib v8 'SAS-library';
```

In the following example, the libref V9Lib accesses only SAS 9 or SAS 8 files.

```
   libname v9lib v9 'SAS-library';
```

In the following example, the libref V6Lib accesses only SAS 6 files.

```
libname v6lib v6 'SAS-library';
```

Determine the Version of SAS Used to Create a SAS File

To determine the version of the SAS engine that was used to create a SAS file, examine the file extension.

Here are the file extensions for files that are created under the Windows operating environment:

Table A14.3 *File Extensions Supported under the Windows Operating Environment*

File Type	SAS 6 File Extension	SAS 9 or SAS 8 File Extension
Data Set	sd2	sas7bdat
Catalog	sc2	sas7bcat
View	sv2	sas7bvew

Concatenate Libraries

In order to expand the scope of SAS file access from a single library to multiple libraries, use library concatenation. With an expanded scope, you can perform operations on either SAS 6 data files or SAS 9 data files that span multiple libraries.

Here is an example of library concatenation:

```
libname v6lib v6 'SAS-library';
libname v9lib v9 'SAS-library';
libname catlib (v9lib v6lib);
```

Note: *SAS-library* must be the physical name that is recognized by the operating environment.

The first LIBNAME statement assigns the libref V6Lib to a SAS library that is accessed using the V6 engine. The V6 engine recognizes only files that are appended with a SAS 6 file extension.

The second LIBNAME statement assigns the libref V9Lib to a SAS library that is accessed using the V9 engine. The V9 engine recognizes only files that are appended with a SAS 9 file extension.

The third LIBNAME statement assigns the libref CATLIB to concatenated SAS libraries that are referenced by the librefs V9Lib and V6Lib. The order of the librefs identifies the sequence in which the libraries are searched. The SAS operation uses the first occurrence of a specified file.

For example, if the same SAS file exists in both SAS libraries and you delete that SAS file, the SAS file in the first library (for example, STOCK.SAS7BDAT in V9Lib) is deleted. If V6Lib precedes V9Lib in the library concatenation statement (for example, STOCK.SD2 in V6Lib), that SAS file is deleted. If the specified SAS file exists in only one SAS library, that SAS file is deleted.

Access SAS Data Sets

Limitations

Accessing data that is stored in a SAS data set is a fundamental operation in SAS. Be aware of any limitations or restrictions when accessing data sets in a cross-version environment. Access to the data files is based on the SAS/CONNECT service that is used, and whether the data files use any new features that are in SAS releases after SAS 6.

SAS 6 Client Accessing a SAS 8 (or later) Server

This table summarizes the limitations of a SAS 6 client that accesses SAS data sets on a SAS 8 (or later) server in a cross-version environment.

Table A14.4 *Limitations for Accessing SAS Data Sets on SAS 8 (or Later) from SAS 6*

SAS/CONNECT Service	SAS 6 Client Connecting to SAS 9 Server	SAS 6 Client Connecting to SAS 8 Server
Remote Library Services	No access is permitted between a SAS 6 client and a SAS 9 server.	If SAS 8 data sets on a SAS 8 server do not implement new features, a SAS 6 client can read, write, or update SAS 8 data sets on a SAS 8 server.
Data Transfer Services	All file formats are automatically converted when uploading or downloading a SAS 6 data set to a SAS 9 or SAS 8 target. If SAS 9 or SAS 8 data sets do not contain new features, they can be downloaded to a SAS 6 target. Truncation rules are applied.	

SAS/CONNECT Service	SAS 6 Client Connecting to SAS 9 Server	SAS 6 Client Connecting to SAS 8 Server
Compute Services	A SAS 6 client can remotely submit a SAS program to a SAS 9 or SAS 8 server. The data sets that are referenced in the remote submit blocks can be SAS 9, SAS 8, or SAS 6 data sets.	

SAS 8 (or Later) Client Accessing a SAS 6 Server

This table summarizes the limitations of a SAS 8 (or later) client that accesses data sets on a SAS 6 server in a cross-version environment.

Table A14.5 *Limitations for Accessing Data Sets on SAS 6 from SAS 8 (or Later)*

SAS/CONNECT Service	SAS 9 Client Connecting to a SAS 6 Server	SAS 8 Client Connecting to a SAS 6 Server
Remote Library Services	No access is permitted between a SAS 9 client and a SAS 6 server.	If SAS 6 data files do not implement new features, a SAS 8 client can read, write, or update SAS 6 data files on a SAS 6 server.
Data Transfer Services	All data formats are automatically converted when uploading or downloading a SAS 6 file to a SAS 9 or SAS 8 target. If SAS 9 or SAS 8 data files do not contain new features, they can be uploaded to a SAS 6 target. Truncation rules are applied.	
Compute Services	A SAS 9 or SAS 8 client can remote submit a SAS program to a SAS 6 server. The data files that are referenced in the remote submit blocks can be formatted only as SAS 6 files.	

Access SAS Views

Limitations

There are limitations and restrictions when accessing SAS views in a cross-version environment. Here are the types of SAS views:

- DATA step
- PROC SQL
- SAS/ACCESS

Note: SAS/CONNECT uses the data that the SAS view references, but not the SAS view itself.

SAS 6 Client Accessing a SAS 8 (or Later) Server

This table summarizes the limitations of a SAS 6 client that accesses SAS views on a SAS 8 (or later) server in a cross-version environment.

Table A14.6 *Limitations for Accessing SAS Views on SAS 8 (or Later) from SAS 6*

SAS/CONNECT Service	SAS 6 Client Connecting to SAS 9 Server	SAS 6 Client Connecting to SAS 8 Server
Remote Library Services	No access is permitted between a SAS 6 client and a SAS 9 server.	For SAS 8 DATA step views, the SAS 6 client has only Read access. For SAS 8 SAS/ACCESS views, the SAS 6 client has Read, Write, and Update access.
Data Transfer Services	For PROC SQL views, a SAS 6 client can upload a PROC SQL view between a SAS 9 or SAS 8 server by using the INLIB= option to specify the library that is associated with the view to transfer. The DATA= option can be used, but a data set will be created.	
Compute Services	For SAS views, a Version 6 client can remote submit a SAS program that references SAS views to a SAS 9 or SAS 8 server. The SAS views that are referenced in	

SAS/CONNECT Service	SAS 6 Client Connecting to SAS 9 Server	SAS 6 Client Connecting to SAS 8 Server
	remote submit blocks can be SAS 9, SAS 8, or SAS 6 data files.	

SAS 8 (or Later) Client Accessing a SAS 6 Server

This table summarizes the limitations of a SAS 8 (or later) client that accesses SAS views on a SAS 6 server in a cross-version environment.

Table A14.7 *Limitations for Accessing SAS Views on SAS 6 from SAS 8 (or Later)*

SAS/CONNECT Service	SAS 9 Client Connecting to a SAS 6 Server	SAS 8 Client Connecting to a SAS 6 Server
Remote Library Services	No access is permitted between a SAS 9 client and a SAS 6 server.	For SAS 6 DATA step views and SAS 6 PROC SQL views, if the view is processed at the server (RMTVIEW=YES in the LIBNAME statement), the SAS 8 client has Read access only for DATA step views.
Data Transfer Services	A SAS 9 or SAS 8 client can upload data that is associated with a SAS view to a SAS 6 server. Names of files that are transferred to a SAS 6 server are truncated, following truncation rules.	
Compute Services	A SAS 9 or SAS 8 client can remotely submit a SAS program that references SAS 6 views to a SAS 6 server.	

Access Catalogs

Limitations

There are limitations and restrictions when accessing catalogs in a cross-version environment.

CAUTION

A SAS 9 or SAS 8 SAS session cannot read SAS 6 catalogs on AIX RS/6000. Use the CPORT and CIMPORT procedures to migrate SAS 6 catalogs into a SAS 9 or SAS 8 environment on AIX.

SAS 8 (or later) catalog entry types (alphabetized horizontally) that are compatible with SAS 6 include:

AFCBT	AFGO	DEVMAP
FONT	FONTLIST	KEYMAP
KEYS	LOG	OUTPUT
SOURCE	TEMPLATE	TRANTAB

SAS 6 Client Accessing a SAS 8 (or Later) Server

This table summarizes the limitations of a SAS 6 client that accesses catalogs on a SAS 8 (or later) server in a cross-version environment.

Table A14.8 *Limitations for Accessing Catalogs on SAS 8 (or Later) from SAS 6*

SAS/CONNECT Service	SAS 6 Client Connecting to SAS 9 Server	SAS 6 Client Connecting to SAS 9 Server
Remote Library Services	No access is permitted between a SAS 9 client and a SAS 6 server.	A SAS 6 client can read a SAS 6 catalog on a SAS 8 server. A SAS 6 client can read, write, and update a SAS 8 catalog that does not contain new features.
Data Transfer Services	A SAS 6 client can upload a SAS 6 catalog to a SAS 9 or SAS 8 server. The uploaded catalog is converted to SAS 9 or SAS 8 format. A SAS 6 client can download a SAS 9 or SAS 8 catalog if the entry type does not contain new features.	
Compute Services	A SAS 6 client can remotely submit a SAS program that references a SAS catalog to a SAS 9 or SAS 8 server.	

SAS 8 (or Later) Client Accessing a SAS 6 Server

This table summarizes the limitations of a SAS 8 (or later) client that accesses catalogs on a SAS 6 server in a cross-version environment.

Table A14.9 *Limitations for Accessing Catalogs on SAS 6 from SAS 8 (or Later)*

SAS/CONNECT Service	SAS 9 Client Connecting to a SAS 6 Server	SAS 8 Client Connecting to a SAS 6 Server
Remote Library Services	No access is permitted between a SAS 9 client and a SAS 6 server.	A SAS 8 client can read from and write to a SAS 6 catalog on a SAS 6 server. A SAS 8 client can write a SAS 6 catalog from one SAS 6 library to another SAS 6 library by using PROC COPY.
Data Transfer Services	A SAS 9 or SAS 8 client can download a Version 6 catalog from a SAS 6 server. A SAS 9 or SAS 8 server can upload a SAS 6 catalog from a SAS 9 or Version 8 server if the entry type does not contain new features. A SAS 9 or SAS 8 client cannot create a SAS 6 catalog entry by using PROC UPLOAD.	
Compute Services	A SAS 9 or SAS 8 client can remotely submit a SAS program that references a SAS catalog to a SAS 6 server.	

File Format Translation Algorithms

SAS 6 Translation

In SAS 6, translation occurs twice for each data transfer between a client and a server that run on computers whose architectures are incompatible.

1 The data is translated from the source computer's native file format to transport format.

2 The data that is represented in transport format is translated to the target computer's native file format.

SAS 8 (and Later) Translation

In SAS 8 and later releases of SAS, translation occurs only once for each data transfer between a client and a server that run on computers whose architectures are incompatible. SAS/CONNECT dynamically translates incompatible file formats for each file upload or file download transaction, bypassing the explicit creation of a transport file.