SAS Publishing

SAS/CONNECT® 9.1

User's Guide

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *SAS/CONNECT® 9.1: User's Guide*. Cary, NC: SAS Institute Inc.

SAS/CONNECT® 9.1: User's Guide

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

1st printing, January 2004

Contents

What's New

Overview

New and enhanced features for SAS/CONNECT improve ease of use and SAS performance:

- MP CONNECT technology encompasses parallel processing on SMP (Symmetric Multi-Processor) hardware as well as on machines across a network via the asynchronous RSUBMIT statement or command. MP CONNECT now supports pipeline parallelism.
- SAS/CONNECT libref inheritance eliminates the need to duplicate data for use in multiple SAS sessions.
- The same command that is used to start a client session can also be used to start a server session by assigning the value **`!sascmd`** to the SASCMD= option in the SIGNON and RSUBMIT statements and to the SASCMD= global system option.
- To ensure information privacy, SAS/CONNECT now supports the network security protocol Secure Sockets Layer (SSL), which encrypts client/server data transfers.
- The %SYSLPUT macro statement now enables you to create a macro variable in a specific server session.
- A new SASESOCK engine in the LIBNAME statement is available for SAS/ CONNECT applications that implement MP CONNECT with piping.
- Encoded passwords are supported in the RSUBMIT statement, the SIGNON statement, and in script files that contain passwords.
- The SIGNON statement supports these new options: SIGNONWAIT=, TBUFSIZE=, SERVER=, and SERVERV=.
- Compute Services (RSUBMIT statement) supports the new options SIGNONWAIT= and CPERSIST=.
- New SAS global system options are SIGNONWAIT= and SYSRPUTSYNC=.
- SAS/CONNECT server initialization errors are written to the SAS console log.

Note:

- This section describes the features of SAS/CONNECT that are new or enhanced since SAS 8.2.

- z/OS is the successor to the OS/390 operating system. SAS/CONNECT 9.1 is supported under both OS/390 and z/OS operating systems and, throughout this document, any reference to z/OS also applies to OS/390, unless otherwise stated.

△

Details

Signon

- SAS/CONNECT libref inheritance eliminates the need to duplicate data for use in multiple SAS sessions. Server sessions can inherit client-defined librefs, allowing multiple sessions to read from and write to a single SAS library. Libref inheritance is implemented in the For details about the SIGNON statement, see “SIGNON Statement and Command” on page 63.
- The same command that is used to start a client session can also be used to start a server session by assigning the value **`!sascmd`** to the SASCMD= option in the SIGNON and RSUBMIT statements and the SASCMD global system option. For details about the SIGNON statement, see “SIGNON Statement and Command” on page 63.
- The SIGNONWAIT= option in the SIGNON statement and in the RSUBMIT statement specifies whether implicit signons within an RSUBMIT are executed synchronously or asynchronously during a SAS session. Synchronous processing means that an RSUBMIT to a server session must be completed before control is returned to the client session.For details about the SIGNONWAIT= option, see “RSUBMIT Statement and Command” on page 131.

  SIGNON= is now also a global SAS system option. For details, see “SIGNONWAIT System Option” on page 23.
- The SAS 9.1 behavior of the *filename* value that is specified in the LOG= (and OUTPUT=) options in the SIGNON statement is different than in previous releases of SAS. The most recent RSUBMIT to *filename* overwrites the previous contents of *filename*. Now, instead of overwriting the file, RSUBMIT appends to the file. For details about the LOG= option, see “RSUBMIT Statement and Command” on page 131
- The TBUFSIZE= global system option is now also an option in the SIGNON statement. TBUFSIZE= option in the SIGNON statement 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. For details, see “SIGNON Statement and Command” on page 63.
- The SERVER option in the SIGNON statement is used to specify a CONNECT server definition that has been defined in a SAS Metadata Repository. For details, see “SIGNON Statement and Command” on page 63.
- The SERVERV option in the SIGNON statement is used to view the values that are configured in the CONNECT server definition. For details, see “SIGNON Statement and Command” on page 63.
- In order to recover from initialization failures, you need to view the content of the SAS console log . The location of the log varies according to the operating environment that the server session runs under. For details about the console log, see “SAS/CONNECT Server Session Initialization Errors” on page 100.

Compute Services

- MP CONNECT technology encompasses parallel processing on SMP (Symmetric Multi-Processor) hardware as well as on machines across a network via the asynchronous RSUBMIT statement or command. For an overview of MP CONNECT, see "MP CONNECT" on page 107.
- MP CONNECT now supports pipeline parallelism, which allows multiple SAS DATA steps or procedures to overlap execution by piping the output from one process as the input into the next process in a pipeline. Piping improves performance and reduces the demand for disk space. For details about piping, see "Pipeline Parallelism" on page 109.
- SAS/CONNECT libref inheritance eliminates the need to duplicate data for use in multiple SAS sessions. Server sessions can inherit client-defined librefs, allowing multiple sessions to read from and write to a single SAS library. Libref inheritance is implemented in the For details about the RSUBMIT statement, see "RSUBMIT Statement and Command" on page 131.
- The same command that is used to start a client session can also be used to start a server session by assigning the value `!sascmd` to the SASCMD= option in the RSUBMIT and SIGNON statements and the SASCMD global system option. For details about the RSUBMIT statement, see "RSUBMIT Statement and Command" on page 131.
- The %SYSLPUT statement is used to create a macro variable in a server session. Now you can identify the specific server session in which a macro should be created, which is necessary because multiple server sessions can result from executing asynchronous remote submits. The /REMOTE= option that is specified in the %SYSLPUT macro statement directs the macro definition to a specific server session. For details about the %SYSLPUT statement, see "%SYSLPUT Statement" on page 148.
- The SIGNONWAIT= option in the RSUBMIT statement and in the SIGNON statement specifies whether implicit signons within an RSUBMIT are executed synchronously or asynchronously during a SAS session. Synchronous processing means that an RSUBMIT to a server session must be completed before control is returned to the client session.For details about the SIGNONWAIT= option, see "RSUBMIT Statement and Command" on page 131.

  SIGNON= is now also a global SAS system option. For details, see "SIGNONWAIT System Option" on page 23.
- The SAS 9.1 behavior of the *filename* value that is specified in the LOG= (and OUTPUT=) options in the RSUBMIT statement is different than in previous releases of SAS. The most recent RSUBMIT to *filename* overwrites the previous contents of *filename*. Now, instead of overwriting the file, RSUBMIT appends to the file. For details about the LOG= option, see "RSUBMIT Statement and Command" on page 131
- The CPERSIST option in the RSUBMIT statement is now also a global system option. CPERSIST specifies whether a connection between a client and a server persists (continues) after the RSUBMIT has completed. For details, see "CONNECTPERSIST System Option" on page 15.
- The SYSRPUTSYNC option in the RSUBMIT statement is now also a global system option. SYSRPUTSYNC sets %SYSRPUT macro variables in the client session when the %SYSRPUT statements are executed rather than when a synchronization point is encountered. For details, see "SYSRPUTSYNC System Option" on page 24.

Remote Library Services

- An explicit port number may now be specified in the SERVER= option in the LIBNAME statement if you use the TCP/IP access method and sign on to a spawner by using an explicit port specification instead of using the default port. For details about the SERVER= option in the LIBNAME statement, see "LIBNAME Statement" on page 195.
- SAS 9 Remote Library Services does *not* permit a SAS 9 client or server to connect to a SAS 6 client or server. Use PROC UPLOAD and PROC DOWNLOAD to connect clients and servers that run SAS 9 and SAS 6. For details, see information about cross-version issues in the *SAS/SHARE User's Guide*.
- The LIBNAME statement for the 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. For details, see "LIBNAME Statement, SASESOCK Engine" on page 199.

Data Transfer Services

Improvements to the file-compression algorithm have significantly reduced the time required for large data transfers using SAS/CONNECT. For details about the algorithm, see "File Transfer Performance" on page 218.

Security

- Secure Sockets Layer (SSL) is a protocol that provides network security and privacy for SAS/CONNECT client/server transfers. To use SSL, you must first install and configure it. To apply SSL, you must set the appropriate SAS system options. For details about installing and configuring SSL, see Chapter 30, "Secure Sockets Layer (SSL)," on page 283. For details about the SAS system options, see Chapter 4, "Secure Sockets Layer (SSL) Options," on page 33.

  SSL is supported only under the UNIX and Windows operating environments.
- Encoded passwords are supported in the RSUBMIT statement, the SIGNON statement, and in script files that contain passwords. For details about specifying an encoded password in the RSUBMIT statement, see "RSUBMIT Statement and Command" on page 131. For details about specifying an encoded password in the SIGNON statement, see "SIGNON Statement and Command" on page 63. For details about specifying an encoded password in a script file, see "Using Passwords in a Script File" on page 56.

PART 1

What Is SAS/CONNECT?

CHAPTER 1 SAS/CONNECT: Definitions and Services

SAS/CONNECT Terminology

SAS/CONNECT

SAS/CONNECT software is a SAS client/server toolset that provides scalability through parallel SAS processing. SAS/CONNECT provides users and applications developers the ability to manage, access, and process data in a distributed and parallel environment by enabling you to

- Achieve SAS interoperability across architectures and SAS releases
  - directly process to a remote data source and get results back locally
  - transfer disk copies of data
  - develop local graphical user interfaces that process remote data sources.
- Develop scalable SAS solutions
  - run multiple independent processes asynchronously and coordinate the results from each task execution in your client SAS session
  - scale up to fully use the capabilities of SMP hardware, and scale out to fully use the features of distributed processors
  - use pipeline processing (TCP/IP ports) to run multiple dependent processes asynchronously
  - collect the resources of multiple machines that work in parallel, which produces a powerful, yet inexpensive processing solution.
- 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.

The Client/Server Relationship

SAS/CONNECT links a SAS client session to a SAS server session. The terms SAS/CONNECT *client* and *server* depict a relationship between two SAS sessions.

The client session is the initial SAS session that creates and manages one or more server sessions. The server sessions can run either on the same machine as the client (for example, an SMP machine) or on a remote machine across a network.

Single-User Server

SAS/CONNECT provides the single-user server functionality for Remote Library Services (RLS), which

- provides transparent access to remote data.
- gives single-user access to a dedicated server.
- enables full, unrestricted access to DBMS data via a SAS/ACCESS engine.
- enables you to connect to the server by using a SIGNON statement and a LIBNAME statement that specifies the REMOTE engine.

  ```
  SIGNON server-ID;
  LIBNAME libref REMOTE 'datalib' SERVER=server-ID;
  ```

  The LIBNAME statement implicitly starts the single-user server.

Multi-User Server

SAS/SHARE provides the multi-user server functionality for Remote Library Services (RLS), which

- gives concurrent, multi-user access to a server

  Note: The ability to access DMBS data through a multi-user server is controlled by a specific SAS/ACCESS engine. △

- is explicitly started and controlled by a system administrator.

  ```
  PROC SERVER server=server-ID;
  ```

- enables you to connect to the server by using a LIBNAME statement that specifies the REMOTE engine.

  ```
  LIBNAME libref REMOTE 'datalib' SERVER=server-ID;
  ```

  The LIBNAME statement causes a connection to a pre-existing server.

Communications Access Method

A *communications access method* is the interface between SAS/CONNECT and the network protocol that you use to connect two SAS sessions. You must specify a communications access method for SAS/CONNECT.

TCP/IP is the supported access method on all SAS 9.1 operating environments. The XMS access method is used to connect client and server sessions that both run under z/OS.

Before any meaningful work can be accomplished between a client and a server, the access method must be configured in the client and the server environments. For details, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Programming Services

Compute Services and MP CONNECT

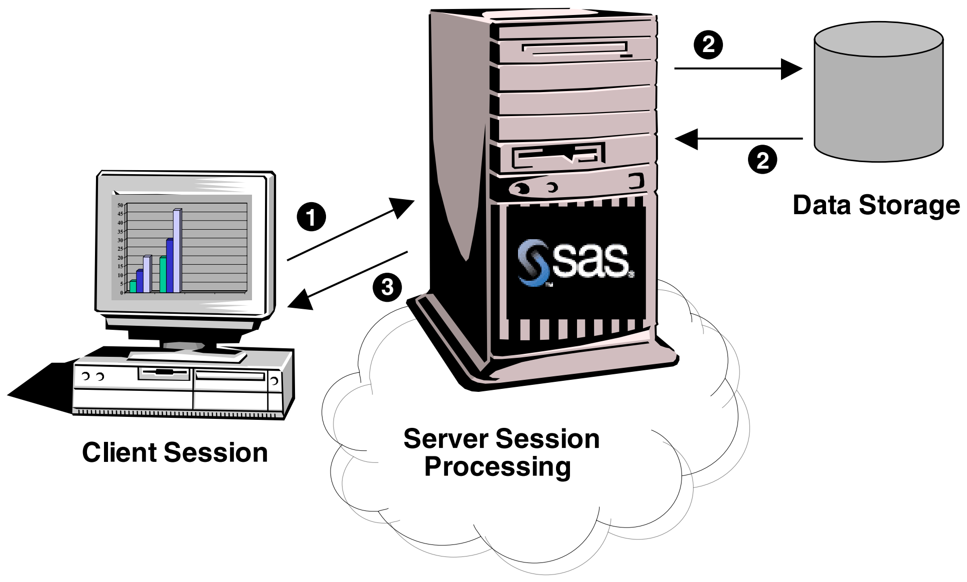
Compute Services That Use RSUBMIT

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. 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 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.

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

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

Figure 1.1 Model of Compute Services



Legend

❶ Send SAS statements to the server session for processing

❷ Use server data in the server session

❸ Receive results at the client

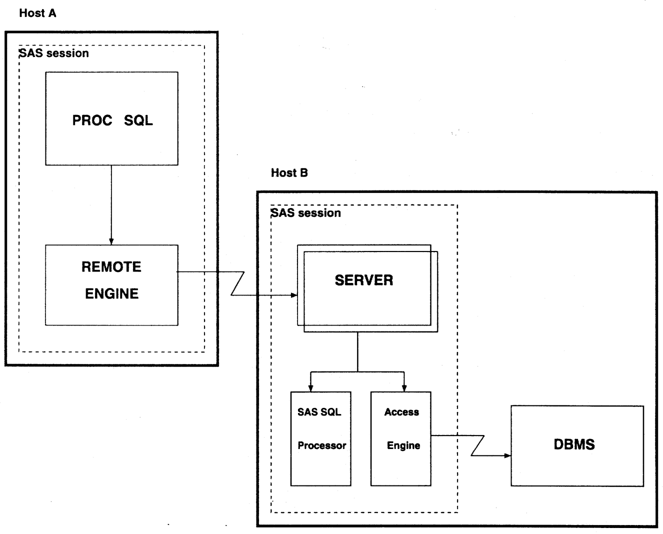
Compute Services enables you to:

- 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 machine, 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 machines.
  - execute against the remote copy of the data.
  - remote submit macro steps 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



RSPT enables you to:

□ pass SQL statements to a remote DBMS in order to select data or to execute statements in order to modify, manipulate, and manage data. This includes creating DBMS views.

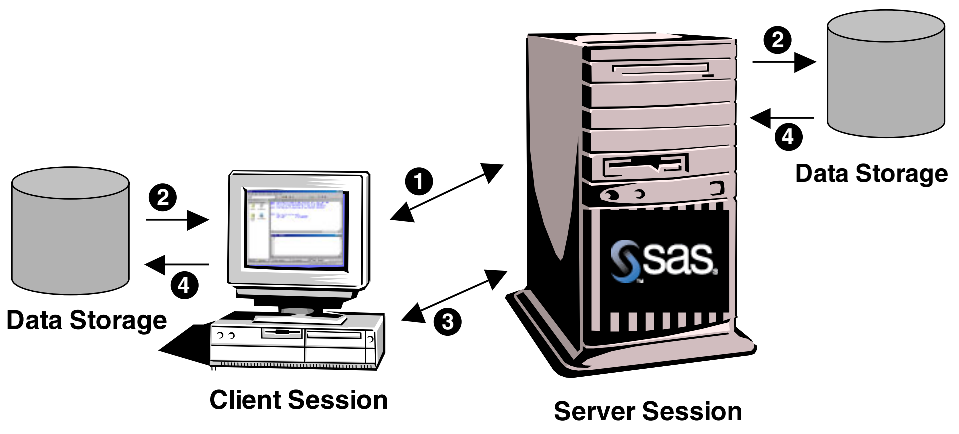
□ pass SQL statements to SAS SQL in order to select data or to execute statements in order to modify, manipulate, and manage data. This includes creating SAS SQL views.

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.

Data Transfer Services

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

Figure 1.3 Model of Data Transfer Services (UPLOAD and DOWNLOAD)



Legend

❶ **Request upload of data records from the client to the server**

❷ **Data is copied from the client and written to disk on the server**

❸ **Request download of data records from the server to the client**

❹ **Data is copied from the server and written to disk on the client**

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

Note: External files can be transferred in either text or binary format. △

The data transfer capabilities enable you to:

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

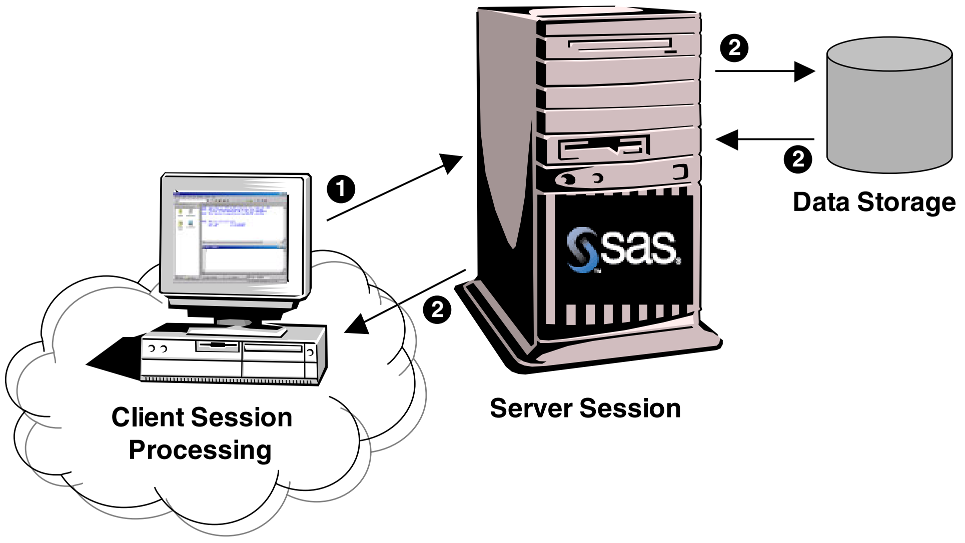
- □ automate both data or application distribution and centralized data collection.
- □ distribute files from one workstation by uploading to a server and downloading to other workstations that need the files.

□ move SAS files between releases of SAS as well as across operating environments.

Remote Library Services

Remote Library Services (RLS) provides transparent access to SAS data that is located on a remote machine. 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 1.4 Model of RLS Processing



Legend

❶ **Client processing requests records from the server or requests records to be written to the server**

❷ **Data records written to the server or sent to the client for 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:

□ transparent access to SAS data that is located on a remote machine.

□ 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 machine.

P A R T 2

SAS/CONNECT SAS System Options

CHAPTER

2
SAS/CONNECT General SAS System Options

AUTOSIGNON System Option

Automatically signs on to the server when the client issues a remote submit request for server processing.

Client: Optional

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

Default: NOAUTOSIGNON

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

AUTOSIGNON | NOAUTOSIGNON

Syntax Description

AUTOSIGNON
automatically signs the client on to the server session when an RSUBMIT command or statement executes. When RSUBMIT ends, an automatic signoff occurs, by default.

NOAUTOSIGNON
does not automatically sign the client on to the server session when an RSUBMIT command or statement executes. In order to connect the client and server sessions, you must explicitly issue the SIGNON command or statement.

Details

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

Example

For an example of using the autosignon feature with MP CONNECT, see "Example 5: Using MP CONNECT and the WAITFOR Statement" on page 160.

See Also

Statements
"RSUBMIT Statement and Command" on page 131
"SIGNON Statement and Command" on page 63

COMAMID= System Option

Identifies the communications access method for connecting a client and a server across a network.

Client: Required

Server: Required

Default: TCP/IP for OpenVMS Alpha, UNIX, and Windows

Default: XMS for z/OS

Client: Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation

Server: Valid in: configuration file, SAS invocation

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

COMAMID=*access-method-ID*

Syntax Description

access-method-ID
specifies the name of the communications access method that is used by a SAS/CONNECT client to connect to a SAS/CONNECT server across a network.

Examples

At the client, the following OPTIONS statement specifies the TCP/IP access method for connecting to a server.

```
options comamid=tcp;
```

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;
```

See Also

The supported communications access methods by operating environment in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*

CONNECTPERSIST System Option

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

Client: Optional

Alias: CPERSIST

Default: CONNECTPERSIST

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

CONNECTPERSIST | NOCONNECTPERSIST

Syntax Description

CONNECTPERSIST
continues a client/server connection after the RSUBMIT (with or without automatic signon) has completed. A SIGNOFF is not automatically executed.

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

Details

The CONNECTPERSIST option is most useful when automatic signon (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 global system option, CONNECTPERSIST can be set locally as an option in the RSUBMIT statement. The locally set option in the RSUBMIT statement or command takes precedence over the global system option.

See Also System Option

"AUTOSIGNON System Option" on page 13

Statement

"RSUBMIT Statement and Command" on page 131

CONNECTREMOTE= System Option

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

Client: Required

Aliases: CREMOTE=, REMOTE=, PROCESS=

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

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 multi-processor 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 global system option, CONNECTREMOTE= can be set locally as an option in the RSUBMIT and SIGNON statements. The locally set option in an

RSUBMIT or SIGNON statement or command takes precedence over the global system option.

Examples

Example 1: CONNECTREMOTE= in SIGNON At the client, the following OPTIONS statement specifies the TCP/IP access method for connecting to a SAS session on a machine named APEX.

```
options comamid=tcp connectremote=apex;
signon;
```

Alternatively, you can specify the CONNECTREMOTE= option in the SIGNON statement.

```
signon connectremote=apex;
```

After a successful signon, the CONNECTREMOTE= value is updated.

Example 2: CONNECTREMOTE= in RSUBMIT The following OPTIONS statement specifies the TCP/IP access method for connecting to a SAS session on the machine named APEX, which connects to the session ID of the OpenVMS server that statements are remote submitted to.

```
options comamid=tcp connectremote=apex;
rsubmit;
   statements for OpenVMS server
endrsubmit;
```

The following OPTIONS statement specifies the TCP/IP access method and the macro variable HOST1, which contains the IP address of a UNIX server that the statements are remote submitted to.

```
%let host1=IP-address;
options comamid=tcp connectremote=host1;
rsubmit;
   statements for UNIX server
endrsubmit;
```

Alternatively, you can specify the session ID directly in the RSUBMIT statement.

```
rsubmit apex;
   statements for OpenVMS server
endrsubmit;

%let host1=IP-address;
rsubmit host1;
   statements for UNIX server
endrsubmit;
```

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

See Also

Statement

“RSUBMIT Statement and Command” on page 131

“SIGNON Statement and Command” on page 63

CONNECTSTATUS System Option

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

Client: Optional

Aliases: CSTATUS, STATUS

Default: CONNECTSTATUS

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

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, locally, in subsequent PROC UPLOAD, PROC DOWNLOAD, RSUBMIT, and SIGNON statements.

For asynchronous processing (NOCONNECTWAIT), the CONNECTSTATUS global 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 about "Transfer Status Window" on page 219

Statements

"RSUBMIT Statement and Command" on page 131

"SIGNON Statement and Command" on page 63

Procedures

"Syntax for the UPLOAD Procedure" on page 224

"Syntax for the DOWNLOAD Procedure" on page 240

CONNECTWAIT System Option

Specifies whether remote submits are executed synchronously or asynchronously.

Client: Optional

Alias: CWAIT

Default: CONNECTWAIT

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

CONNECTWAIT | NOCONNECTWAIT

Syntax Description

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

NOCONNECTWAIT
specifies that RSUBMITs 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 global system option specifies whether remote submits are executed synchronously. The default setting can be overridden by setting the CONNECTWAIT= option, locally, in the SIGNON statement or in subsequent RSUBMIT statements. The locally set option in the RSUBMIT or SIGNON statement or command takes precedence over the global 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 Statement and Command” on page 131

“SIGNON Statement and Command” on page 63

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 multi-processor (SMP) machine.

Client: Optional

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

OpenVMS Alpha, UNIX, Windows

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

z/OS

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

Syntax Description

SASCMD= <"*SAS-command* <*SAS-system-options*>" | "!sascmd *SAS-system-options*">
under the OpenVMS Alpha, UNIX, and Windows operating environments, this command starts a server session on a multi-processor machine. 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.

SASCMD= <":*SAS–system–options*" | "!sascmd *SAS-system-options*">
under the z/OS operating environment, this command starts a server session on a multi-processor machine, and passes values for the following SAS system options to the server session: DMR, COMAMID=, REMOTE=, SASHELP=, SASMSG=, SASAUTOS=, and CONFIG=. You may 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.

Details

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

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

Examples

The following OPTIONS statement invokes a SAS session.

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

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

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

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

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

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" %1 %2 %3 %4 %5 %6 %7 %8 %9
```

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

See Also

Statements

"RSUBMIT Statement and Command" on page 131

"SIGNON Statement and Command" on page 63

SASFRSCR System Option

Is a read-only option that contains the fileref that is generated by the SASSCRIPT= option.

Client: 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.

For more information, see “SASSCRIPT= System Option” on page 22.

SASSCRIPT= System Option

Specifies one or more storage locations for SAS/CONNECT script files.

Client: Optional

Default: Varies by operating environment.

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SASSCRIPT= ’*dir-name*’ | (’*dir-name-1*’, ... ,’*dir-name-n*’)

Syntax Description

’dir-name’

specifies the name of one or more directories that contain SAS/CONNECT script files. Table 2.1 on page 22 shows the default storage locations for these files by operating environment.

Table 2.1 Default Storage Location for SAS/CONNECT Script Files

Operating Environment	Script File Location
OpenVMS Alpha	SAS$ROOT:[TOOLS]
z/OS	&prefix.CTMISC

Operating Environment	Script File Location
UNIX	!sasroot/misc/connect
Windows	!SASROOT\CONNECT\SASLINK

Details

If the CSCRIPT= option is specified in the SIGNON statement and the specified script file is *not* located in the current directory, the location specified in the SASSCRIPT= option is checked.

The SASSCRIPT= option also enables you to identify the location of a script file that is specified in a SAS/CONNECT SERVER definition that is stored in the metadata repository.

The metadata system options are: METAAUTORESOURCES=, METACONNECT=, METAID=, METAPASS=, METAPORT=, METAPROFILE=, METAPROTOCOL=, METAREPOSITORY=, METASERVER=, and METAUSER=.

Example

In the following OPTIONS statement, the SASSCRIPT= option is used to override the default storage location for script files under the Windows operating environment.

```
options sasscript= "c:\my\favorite\scripts";
```

After an alternative storage location has been defined, a script might be specified as follows:

```
signon remhost script="myscr.scr";
```

Because **`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.

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. △

See Also

Statements

“RSUBMIT Statement and Command” on page 131

“SIGNON Statement and Command” on page 63

System Options

Metadata Repository System Options in *SAS Language Reference: Dictionary*

SIGNONWAIT System Option

Specifies whether a SAS/CONNECT SIGNON should be executed asynchronously or synchronously

Client: Optional

Aliases: CONNECTSWAIT, SWAIT

Default: SIGNONWAIT

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SIGNONWAIT | NOSIGNONWAIT

Syntax Description

SIGNONWAIT
specifies that a SAS/CONNECT SIGNON will execute synchronously. *Synchronous processing* means that a signon to a server session must complete before control is returned to the client session.

NOSIGNONWAIT
specifies that a SAS/CONNECT SIGNON will execute asynchronously. *Asynchronous processing* permits signons to multiple server sessions to execute in parallel. Control is returned to the client session immediately after a signon 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 remote submitting units of work to a server session, as soon as signon 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 global system option, SIGNONWAIT can be set locally as an option in the RSUBMIT and SIGNON statements. The locally set option in the RSUBMIT or SIGNON statement or command takes precedence over the global system option.

See Also

Statements

“RSUBMIT Statement and Command” on page 131

“SIGNON Statement and Command” on page 63

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

Alias: CSYSRPUTSYNC

Default: NOSYSRPUTSYNC

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SYSRPUTSYNC | NOSYSRPUTSYNC

Syntax Description

SYSRPUTSYNC
forces the macro variables to be set when %SYSRPUT executes.

NOSYSRPUTSYNC
sets macro variables in the client session when a synchronization point is encountered.

Details

This option is valid only when executing an asynchronous RSUBMIT, which occurs when the NOCONNECTWAIT option is in effect. If the NOCONNECTWAIT option is not specified, the SYSRPUTSYNC option is ignored.

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

See Also

Conceptual information about "Synchronization Points" on page 146

Statements

"RSUBMIT Statement and Command" on page 131

"SIGNON Statement and Command" on page 63

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

Default: 32768 (the value of 0 is converted to 32768)

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

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 *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Table 2.2 Summary of Attributes for the TBUFSIZE= and TCPMSGLEN= Options

System Option	Default Buffer Size	Controlling SAS Layer	Purpose of Buffer
TBUFSIZE	32K	SAS Application	SAS/CONNECT uses the buffer to transfer data to the communications layer.
TCPMSGLEN	32K for OpenVMS Alpha, OS/390, and UNIX; 16K for Windows	SAS Communications	The TCP/IP access method uses the buffer to transfer data to a client or a server.

The SAS application layer:

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 whose size is defined in the TBUFSIZE= option to the communications layer.

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:

1 receives a buffer from the SAS application layer
2 sends a buffer whose size is defined in the TCPMSGLEN= option to the client or to the server.

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 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 global system option, TBUFSIZE= can be set locally as an option in the SIGNON statement. The locally set option in the SIGNON statement or command takes precedence over the global system option.

Example

In the following OPTIONS statement, the TBUFSIZE= option is used to increase the buffer size from 32K (the default) to 64K:

```
options tbufsize=65536;
signon;
```

Alternatively, you can specify **`tbufsize=64k`**.

See Also

System Option

TCPMSGLEN system option that is used by the TCP/IP access method for the appropriate operating environment in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*

Statement

"SIGNON Statement and Command" on page 63

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

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.

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

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.

CHAPTER

3

SAS/SECURE and SASProprietary Encryption Services

NETENCRYPT System Option

Specifies whether client/server data transfers are encrypted.

Client: Optional

Server: Optional

Default: NONETENCRYPT

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

See also: NETENCRYPTALGORITHM

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

NETENCRYPT | NONETENCRYPT

Syntax Description

NETENCRYPT
specifies that encryption *is* required.

NONETENCRYPT
specifies that encryption is *not* required, but is optional.

Details

The default for this option specifies that encryption *is* used if the NETENCRYPTALGORITHM option is set and if both the client and the server are capable of encryption. If encryption algorithms are specified but either the client or the server is incapable of encryption, then encryption is *not* performed.

Encryption might *not* be supported at the client or at the server if

- You are using a release of SAS (prior to Version 8) that does not support encryption.
- Your site (the client or the machine where the spawner is running) does not have a security software product installed.
- You specified encryption algorithms that are incompatible in SAS sessions on the client and the server.

NETENCRYPTALGORITHM= System Option

Specifies the algorithm(s) to be used for encrypted client/server data transfers.

Client: Optional

Server: Required

Alias: NETENCRALG=

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

See also: NETENCRYPT

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

NETENCRYPTALGORITHM=*algorithm* | *("algorithm-1"... "algorithm-n")*

Syntax Description

algorithm* | *("algorithm–1"... "algorithm-n")
specifies the algorithm(s) that can be used for encrypting data that is transferred between a client and a server across a network. When you specify two or more encryption algorithms, use a space or a comma to separate them, and enclose the algorithms in parentheses.

The following algorithms may be used:

- RC2
- RC4
- DES
- TripleDES
- SASProprietary
- SSL.

Details

The NETENCRYPTALGORITHM= option *must* be specified in the server session.

Use this option to specify one or more encryption algorithms that you want to use to protect the data that is transferred across the network. If more than one algorithm is specified, the client session negotiates the first specified algorithm with the server session. If the server session does not support that algorithm, the second algorithm is negotiated, and so on.

If either the client or the server session specifies the NETENCRYPT option (which makes encryption mandatory) but a common encryption algorithm cannot be negotiated, the client *cannot* connect to the server.

If the NETENCRYPTALGORITHM= option is specified in the server session only, then the server's values are used to negotiate the algorithm selection. If the client session supports only one of multiple algorithms that are specified in the server session, the client *can* connect to the server.

There is an interaction between either NETENCRYPT or NONETENCRYPT and NETENCRYPTALGORITHM.

Table 3.1 Client/Server Connection Outcomes

Server Settings	**Client Settings**	**Connection Outcome**
NONETENCRYPT NETENCRYPTALGORITHM=*algorithm*	No settings	If the client is capable of encryption, the client/server connection will be encrypted. Otherwise, the connection will not be encrypted.
NETENCRYPT NETENCRYPTALGORITHM=*algorithm*	No settings	If the client is capable of encryption, the client/server connection will be encrypted. Otherwise, the client/server connection will fail.
No settings	NONETENCRYPT NETENCRYPTALGORITHM=*algorithm*	A client/server connection will not be encrypted.
No settings	NETENCRYPT NETENCRYPTALGORITHM=*algorithm*	A client/server connection will fail.
NETENCRYPT or NONETENCRYPT NETENCRYPTALGORITHM=*algorithm-a*	NETENCRYPTALGORITHM=*algorithm-b*	Regardless of whether NETENCRYPT or NONETENCRYPT is specified, a client/ server connection will fail.

Example

In the following example, the client and the server specify different values for the NETENCRYPTALGORITHM= option.

The client specifies two algorithms in the following OPTIONS statement:

```
options netencryptalgorithm=(rc2 tripledes);
```

The server specifies three algorithms and requires encryption in the following OPTIONS statement:

```
options netencrypt netencryptalgorithm=(ssl des tripledes);
```

The client and the server negotiate an algorithm that they share in common, TripleDES, for encrypting data transfers.

NETENCRYPTKEYLEN= System Option

Specifies the key length to use for encrypting data that is transferred between a client and a server across a network.

Client: Optional

Server: Optional

Alias: NETENCRKEY=

Default: 0

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

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

NETENCRYPTKEYLEN= 0 | 40 | 128

Syntax Description

0
specifies that the maximum key length that is supported at both the client and the server is used.

40
specifies a key length of 40 bits for the RC2 and RC4 algorithms.

128
specifies a key length of 128 bits for the RC2 and RC4 algorithms. If either the client or the server does not support 128-bit encryption, the client cannot connect to the server.

Details

The NETENCRYPTKEYLEN= option supports only the RC2 and RC4 algorithms. The DES, TripleDES, or SSL algorithms are not supported.

Using longer keys consumes more CPU cycles. If you do not need strong encryption, set NETENCRYPTKEYLEN=40 to decrease CPU usage.

CHAPTER

4

Secure Sockets Layer (SSL) Options

SSLCLIENTAUTH System Option

Specifies whether a server should perform client authentication.

Server: Optional
Valid in: configuration file, OPTIONS statement, SAS System Options window, SAS invocation
Operating Environments: UNIX, Windows
Category: Communications: Networking and Encryption
PROC OPTIONS Group= Communications

Syntax

SSLCLIENTAUTH | NOSSLCLIENTAUTH

Syntax Description

SSLCLIENTAUTH
specifies that the server should perform client authentication.

NOSSLCLIENTAUTH
specifies that the server should not perform client authentication.

Details

Server authentication is always performed, but SSLCLIENTAUTH enables a user to control client authentication. This option is valid only when used on a server.

SSLCRLCHECK System Option

Specifies whether a Certificate Revocation List (CRL) is checked when a digital certificate is validated.

Client: Required

Server: Optional

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

Operating Environments: UNIX, Windows

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCRLCHECK | NOSSLCRLCHECK

Syntax Description

SSLCRLCHECK
specifies that CRLs are checked when digital certificates are validated.

NOSSLCRLCHECK
specifies that CRLs are not checked when digital certificates are validated.

Details

A Certificate Revocation List (CRL) is published by a Certificate Authority (CA) and contains a list of revoked digital certificates. The list contains only the revoked digital certificates that were issued by a specific certificate authority. SSLCRLCHECK is required at a server only if client authentication is specified. Because clients always check server digital certificates, this option is required at the client.

SSLCALISTLOC= System Option

Specifies the location of digital certificates for trusted certificate authorities (CA).

Client: Required

Server: Optional

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

Operating Environment: UNIX

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCALISTLOC=“*file–path*”

Syntax Description

“file-path”
specifies the location of a file that contains the digital certificates for the trusted certificate authority (CA).

Details

The SSLCALISTLOC= option identifies the certificate authority that SSL should trust. This option is required at the client because at least one CA must be trusted in order to validate a server’s digital certificate. This option is required at the server only if client authentication is specified.

SSLCERTLOC= System Option

Specifies the location of the digital certificate that is used for authentication.

Client: Optional

Server: Required

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

Operating Environment: UNIX

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCERTLOC=“*file-path*”

Syntax Description

“file-path”
specifies the location of a file that contains a digital certificate.

Details

The SSLCERTLOC= option is required for a server. It is required at the client only if client authentication is specified.

SSLCRLLOC= System Option

Specifies the location of a Certificate Revocation List (CRL).

Client: Optional

Server: Optional

Operating Environment: UNIX

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCRLLOC="*file-path*"

Syntax Description

"file-path"
specifies the location of a file that contains a Certificate Revocation List (CRL).

Details

The SSLCRLLOC= option is required only when the SSLCRLCHECK option is specified.

SSLPVTKEYLOC= System Option

Specifies the location of the private key that corresponds to the digital certificate that was specified by using the SSLCERTLOC= option.

Client: Optional

Server: Required

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

Operating Environment: UNIX

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLPVTKEYLOC="*file-path*"

Syntax Description

"file-path"
specifies the location of the file that contains the private key that corresponds to the digital certificate that was specified by using the SSLCERTLOC= option.

Details

The SSLPVTKEYLOC= option is required at the server only when the SSLCERTLOC= option is specified.

SSLPVTKEYPASS= System Option

Specifies the password that SSL requires for decrypting the private key that is stored in the file that is specified by using the SSLPVTKEYLOC= option.

Client: Optional

Server: Optional

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

Operating Environment: UNIX

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLPVTKEYPASS="*password*"

Syntax Description

"password"
specifies the password that SSL requires in order to decrypt the private key. The private key is stored in the file that is specified by using the SSLPVTKEYLOC= option.

Details

SSLPVTKEYPASS= is required only when the private key is encrypted.

SSLCERTISS= System Option

Specifies the name of the issuer of the digital certificate that SSL should use.

Client: Optional

Server: Optional

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

Operating Environment: Windows

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCERTISS=“*issuer-of-digital-certificate*”

Syntax Description

“issuer-of-digital-certificate”
specifies the name of the issuer of the digital certificate that should be used by SSL.

Details

The SSLCERTISS= option is used with the SSLCERTSERIAL= option to uniquely identify a digital certificate from the Microsoft certificate store.

SSLCERTSERIAL= System Option

Specifies the serial number of the digital certificate that SSL should use.

Client: Optional

Server: Optional

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

Operating Environment: Windows

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCERTSERIAL=“*serial-number*”

Syntax Description

“serial-number”
specifies the serial number of the digital certificate that should be used by SSL.

Details

The SSLCERTSERIAL= option is used with the SSLCERTISS= option to uniquely identify a digital certificate from the Microsoft certificate store.

SSLCERTSUBJ= System Option

Specifies the subject name of the digital certificate that SSL should use.

Client: Optional

Server: Optional

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

Operating Environment: Windows

Category: Communications: Networking and Encryption

PROC OPTIONS Group= Communications

Syntax

SSLCERTSUBJ=“*subject-name*”

Syntax Description

“subject-name”
specifies the subject name of the digital certificate that SSL should use.

Details

The SSLCERTSUBJ= option is used to search for a digital certificate from the Microsoft certificate store.

P A R T 3

Starting and Stopping SAS/CONNECT Software

CHAPTER 5 Starting and Stopping SAS/CONNECT

Starting SAS/CONNECT

Regardless of the interface that is used to start or stop SAS/CONNECT, the basic tasks are the same. For details about the interfaces, see “Interfaces for Starting and Stopping SAS/CONNECT” on page 49.

To start SAS/CONNECT from a SAS/CONNECT client session:

- Specify a communications access method to access the server machine
- Sign on to the server machine.

Specifying a Communications Access Method

To make a SAS/CONNECT client/server connection, in the client session, you must specify TCP/IP as the access method to communicate with the machine that the server session will run on.

Note: TCP/IP is the default communications access method for most operating environments. If the client/server sessions run under the z/OS operating environment, you can specify the XMS access method. △

Example:

```
options comamid=tcp;
```

For details about using communications access methods, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Signing On to the Server

You can use one of the following methods to sign on:

- the same multi-processor machine

  Note: This method is most useful if your client machine is equipped with SMP (Symmetric Multi-Processor) hardware. △

- a spawner
- a Telnet daemon.

Signing On to the Same Multi-Processor Machine

If your client machine is equipped with SMP (Symmetric Multi-Processors), and if you want to run one or more server sessions on your machine, perform these tasks:

1. Specify the server session.
2. Specify the SASCMD command to start SAS.
3. Sign on to the server session.

TCP/IP is used on SMP machines for OpenVMS Alpha, UNIX, and Windows. XMS is used on SMP machines for z/OS only.

Specifying the Server Session

You can specify the server session in an OPTIONS statement:

```
OPTIONS PROCESS=session-ID;
```

or in the SIGNON statement or command:

```
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 multi-processor machine.

Note: PROCESS= and CONNECTREMOTE= can be used interchangeably. For details, see "CONNECTREMOTE= System Option" on page 16. △

For details about the SIGNON= statement, see Chapter 7, “Syntax for the SIGNON and the SIGNOFF Statements and Commands,” on page 63.

Using the SASCMD Option to Specify SAS

Use the SASCMD option to specify the SAS command and any additional options that you want to use to start SAS in a server session on the same multi-processor machine.
The SASCMD option can be specified either in an OPTIONS statement:

```
OPTIONS SASCMD="SAS-command" | "!SASCMD" | "!sascmdv" | "host-command-file";
```

or directly in the SIGNON statement or command:

UNIX Example:

```
SIGNON name SASCMD="!SASCMD -memsize 64M -nonumber";
```

z/OS Example:

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

The -DMR option is automatically appended to the command. If *!SASCMD* or *!SASCMDV* is specified, SAS/CONNECT starts SAS on the server by using the same command that was used to start SAS for the current client session.

Note:

- Under the UNIX and Windows operating environments, !SASCMDV shows the SAS invocation. Under OpenVMS Alpha, !SASCMDV shows a symbol.
- In order to execute additional commands prior to 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.

△

For details, see “SASCMD= System Option” on page 20, and Chapter 7, “Syntax for the SIGNON and the SIGNOFF Statements and Commands,” on page 63.

Examples: Signing On to the Server Session

Example 1:
In the following example, TCP is the access method, SAS1 is the name of the server session, and SAS_START is the command that starts SAS on the same multi-processor machine.

```
options comamid=tcp;
signon sas1 sascmd='sas_start';
```

Example 2:
In the following example, OPTIONS statements set the values for the COMAMID= , SASCMD=, and PROCESS= options. The SASCMD= option identifies SAS_START as the command that starts SAS. The PROCESS= option identifies the server session on the same multi-processor machine. Because the SASCMD= and the PROCESS= options are defined, only a simple SIGNON statement is needed.

```
options comamid=tcp sascmd="sas_start";
options process=sas1;
signon;
```

Signing On by Using a Spawner

1 Ensure that the spawner is running on the server.
2 Specify the server and an optional service.
3 Specify a sign-on script or a user ID and password.

   Note: For Windows, if SSPI (Security Support Provider Interface) is available or the server is not running secured, you do not have to specify a user ID and password. For details, see the topic about SSPI in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*. △

4 Sign on to the server by using a spawner.

Ensuring That the Spawner Is Running on the Server

Before you can access the spawner, the spawner program must be running on the server. For details, see the topic about spawners in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Note: The system administrator for the machine that the spawner runs on must start the spawner. The spawner program on the server cannot be started in the client session. △

Specifying the Server and the Spawner Service

The name of the server can be specified by using an OPTIONS statement:

```
OPTIONS REMOTE=node-name[.service-name | .port-number];
```

or by using the SIGNON statement or command:

```
SIGNON node-name[.service-name | .port-number];
```

node-name is based on 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:

□ the short machine name of the server you are connecting to. This name must be defined in your Domain Name Server (DNS) or in the **HOSTS** file in the operating environment that the client session runs under.
□ a macro variable that contains either the IP address or the name of the server that you are connecting to.

For UNIX and OpenVMS Alpha only:
The process for evaluating *node–name* follows:

1 If *node-name* is a macro variable, the value of the macro variable is passed to the operating environment's GETHOSTBYNAME 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 GETHOSTBYNAME function.
3 If GETHOSTBYNAME fails to resolve *node-name*, an error message is returned and the signon fails.

   Note: The order in which the GETHOSTBYNAME function calls the DNS or searches the HOSTS file to resolve *node-name* varies based on the operating environment implementation. △

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*.

Example 1:

REMHOST is the name of the node on which the spawner runs, and PORT1 is the name of the service that is defined in the client session. The client service PORT1 must be assigned to the same port that the spawner is listening on.

```
signon remhost.port1;
```

Example 2:

In the following example, the macro variable REMHOST is assigned to the fully-qualified name of the machine on which the server runs. This server has a spawner running that is listening on port 5050. The server session that is specified in the SIGNON statement uses the node name REMHOST and the service name 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:

```
%let remhost=pc.rem.us.com 5050;
signon remhost;
```

Specifying a Sign-On Script or a User ID and Password

You can use a sign-on script to sign on to the spawner, or you can sign on to a spawner without a script. If you do not use a sign-on script and if the spawner is running secured, you must supply a user ID and password to sign on to the spawner.

Note: For Windows only: If you use SSPI, supplying a user ID and a password is unnecessary. For details, see the topic about SSPI in *Communications Access Methods for SAS/CONNECT and SAS/SHARE* △

Note: If you connect to a spawner, you can sign on by using a script unless the spawner is started by using the -NOSCRIPT option. If the -NOSCRIPT option is set, you cannot use a script. If there is no script, you do not assign the fileref RLINK in a FILENAME statement. As an alternative, you can specify the -NOSCRIPT option in the SIGNON statement. For information about the spawner that you are connecting to, see the topic about spawners in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*. △

Specifying 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.

To use one of the sample script files that are provided with SAS/CONNECT for signing on and signing off, assign the fileref RLINK to the appropriate script file. As an alternative, you can specify the SCRIPT= option in the SIGNON statement. The script is based on the server that you are connecting to. The location of the sample scripts varies according to operating environment. For default locations, see “Using a Script to Start and Stop SAS/CONNECT” on page 57.

To specify a script, use the FILENAME statement.

UNIX Example:

```
FILENAME RLINK '!sasroot/misc/connect/script-name';
```

script-name specifies the appropriate script file for the server.
The following table lists the scripts that are supplied in SAS software:

Table 5.1 SAS/CONNECT Sign-on Scripts for TCP/IP

Server	Script Name
TSO under OS/390	**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 Alpha	**tcpvms.scr**
UNIX	**tcpunix.scr**
Windows	**tcpwin.scr**

Specifying a User ID and Password

If you are signing on to the spawner without using a script and the spawner is running secured, you must submit the SIGNON statement and provide a user ID and a password in order to log on to the server.

Note: For Windows only: If SSPI is available, you can submit the SIGNON statement without a user ID and password. If SSPI is not available and you are signing on to a secured spawner without using a script, you must provide a user ID and password in order to log on. For details, see the topic about SSPI in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*. △

```
SIGNON USER=user-ID | _PROMPT_ [ PASSWORD=password | _PROMPT_ ];
```

For details, see “SIGNON Statement and Command” on page 63.

Example: Signing On by Using the Spawner

A client connects to a UNIX server by using a spawner and without a script. In the SIGNON statement, RMTHOST.SPAWNER specifies the node RMTHOST and the service SPAWNER. This server specification presumes that a spawner is running on the node RMTHOST, and that the spawner was started by using the service SPAWNER. Specifying USER=_PROMPT_ causes a dialog box to appear so that a user ID and a password can be provided.
Example:

```
options comamid=tcp;
signon rmthost.spawner user=_prompt_;
```

Signing On by Using a Telnet Daemon

1 Specify the server.
2 Specify a sign-on script.

3 Sign on to the server session.

Specifying the Server

The name of the server can be specified either in an OPTIONS statement:

```
OPTIONS REMOTE=node-name;
```

or directly in the SIGNON statement or command:

```
SIGNON node-name;
```

Specifying a Sign-On Script File

When signing on by using the Telnet daemon, specify a sign-on script. The script file is executed by the SIGNON statement or command. By default, the script prompts for user ID and password. For details, see "SIGNON Statement and Command" on page 63.

Example: Signing On to the Server Session

You specify the statements in a client session that runs under UNIX to use the TCP/IP access method to connect to a z/OS server. 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 server RMTNODE, which is specified in the REMOTE= option.

UNIX example:

```
filename rlink '!sasroot/misc/connect/tcptso.scr';
options comamid=tcp remote=rmtnode;
signon;
```

Interfaces for Starting and Stopping SAS/CONNECT

Types of Interfaces for Starting and Stopping SAS/CONNECT

You can use any of these interfaces to start or stop SAS/CONNECT:

- SAS Windowing environment
- SAS Program Editor window
- SAS autoexec file.

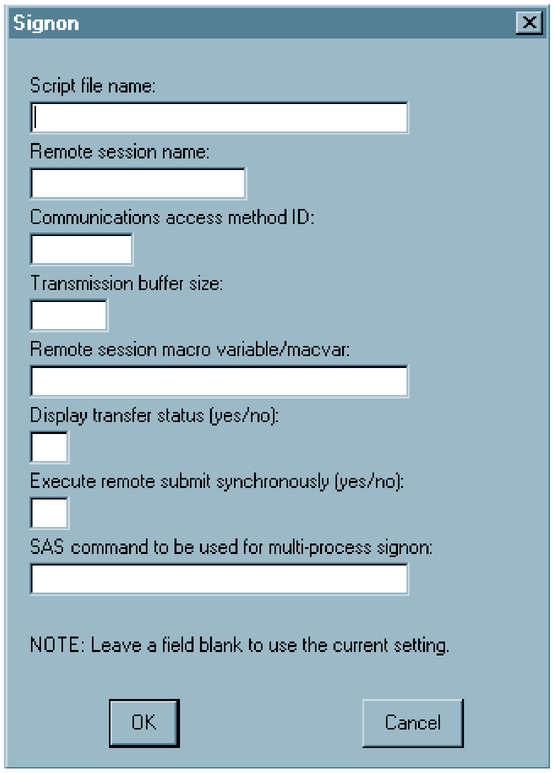
Using the SAS Windowing Environment to Start and Stop SAS/CONNECT

The Signon 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 Signon window.


Signon
Script file name:
Remote session name:
Communications access method ID:
Transmission buffer size:
Remote session macro variable/macvar:
Display transfer status (yes/no):
Execute remote submit synchronously (yes/no):
SAS command to be used for multi-process signon:
OK
Cancel

NOTE: Leave a field blank to use the current setting.

`Script file name:`
If you use the TCP/IP access method and choose to use a script file, type 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, type:

```
pathname/tcptso.scr
```

The default location of the script file varies according to operating environment. For details, see "Using a Script to Start and Stop SAS/CONNECT" on page 57.

`Remote session name:`
Type the name of the session that you are connecting to. For details, see "CONNECTREMOTE= System Option" on page 16.

`Communications access method ID:`
Type the value for the COMAMID= option. For example, for the TCP/IP access method, type:

```
tcp
```

For complete details about access methods, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

`Transmission buffer size:`
Type the value of the buffer size that SAS/CONNECT will use for transferring data. For details, see "TBUFSIZE= System Option" on page 25.

`Remote session macro variable/macvar:`
Type 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= option in the SIGNON statement on page 65.

`Display transfer status (yes/no):`
Type **`yes`** or **`no`** to specify whether the status window is displayed during data transfers. For details, see "CONNECTSTATUS System Option" on page 18.

`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 System Option" on page 19.

`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 multi-processor machine. For details about multi-processing, see "MP CONNECT" on page 107.

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

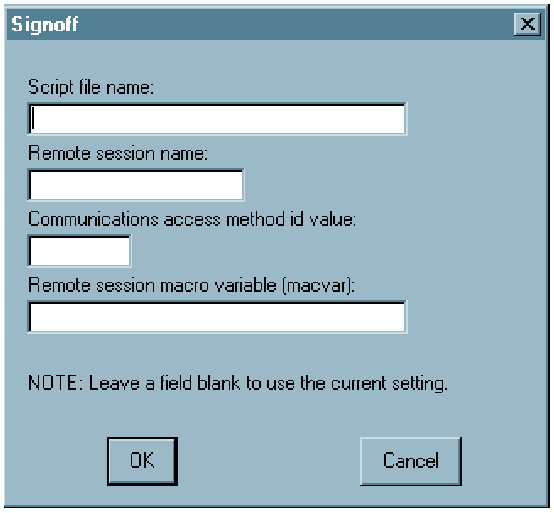
```
filename rlink clear;
```

△

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

The Signoff Window

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


Signoff

Script file name:

Remote session name:

Communications access method id value:

Remote session macro variable (macvar):

NOTE: Leave a field blank to use the current setting.

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 you want to end.

Using the Program Editor Window

The Program Editor Window

1 Type 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;
```

For details about specifying values for these options, see "COMAMID= System Option" on page 14 and "CONNECTREMOTE= System Option" on page 16.

2 Issue the SIGNON command or type 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. △

An example of signing on to a server that is running a spawner program follows:

```
options comamid=communications-method
   remote=nodename.servicename;
signon user=_prompt_;
```

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

Using the Program Editor Window to Sign Off SAS/CONNECT

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

```
signoff cscript='external-file-name-of-script'
```

Note: If you used a script to sign on, the same script can be used to stop the connection. △

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

The sample scripts that are used for automatic signon are used for signing off your server session.

Using the Autoexec File

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. Be sure that it gives the correct file specification for the script that you use to start SAS/CONNECT. For details, see Chapter 7, "Syntax for the SIGNON and the SIGNOFF Statements and Commands," on page 63.

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;
  ```

  By including an OPTIONS statement that contains the COMAMID= and CONNECTREMOTE= system options in the autoexec file, you avoid 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;
```

CHAPTER 6
Using Script Files

Overview to Script Files

A *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.

When to Use a Script

How do you know if you need to write or to modify a script? The need for a script file when using the TCP/IP access method depends on whether you are connecting to a spawner that runs on a server and how that spawner was invoked.

For details about the various access methods, script requirements, and sample script files, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*. Your site might also have sample scripts available from your system administrator.

If the available sample scripts do not meet your requirements, you can write your own script. If you do need to write or to modify a script, review the examples in this chapter, and see Chapter 9, “Script Statements,” on page 87 for descriptions of the script statements that are used in the examples.

Purpose of a Sign-On Script

A 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 communications options for the server session in the SAS command. For the server session, the script sets the COMAMID= and DMR system options.
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. △

Using Passwords in a Script File

Passwords can be specified for a script file in any of these forms:

- □ a clear-text password that is hard-coded 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.

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 the *Base SAS Procedures Guide*.

Example of code that is used to obtain an encoded password:

```
proc PWENCODE in="My2003PW";run;
{sas001}TXkyMDAzUFc=
```

The clear-text password **My2003PW** 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. △

Using a Script to Start and Stop SAS/CONNECT

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:

Windows	!sasroot\CONNECT\SASLINK
z/OS	*prefix*.CTMISC
OpenVMS Alpha	SAS$ROOT:[TOOLS]
UNIX	!sasroot/misc/connect

Note: The term !sasroot is not part of the pathname. It represents the name of the directory where SAS is installed at your site. △

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; for example, TCPTSO.SCR identifies the TCP/IP access method and a TSO server.

Syntax Rules for Script Statements

To write a 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.

- 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 8 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

  n SECONDS;

  where *n* can be any number; this number may include decimal fractions. For example, all of the following time specifications are valid:

  0 SECONDS;

  0.25 SECONDS;

  1 SECOND;

  3.14 SECONDS;

  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.

Writing Simple Scripts for Signing On and Signing Off

Writing Simple Scripts: Overview

When you write or modify existing 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 type to respond to a display should be specified in a TYPE statement. Be sure to note all carriage returns or other special keys.

If z/OS is the server 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, to divide the following TYPE statement:

```
type
"sas options ('dmr comamid=tcp')"
enter;
```

change it to:

```
type "sas options ('dmr comamid=-" enter;
type "tcp')" enter;
```

Note: Do not insert spaces before or after the hyphen. △

Example 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:   1. This script might need      */
  /*             modifications that account  */
  /*             for the local flavor of     */
  /*             your UNIX environment. The  */
  /*             logon procedure should      */
  /*             mimic the tasks that you    */
  /*             execute when                */
  /*             connecting to the same      */
  /*             UNIX operating environment. */
  /*                                        */
  /*          2. You must have specified     */
  /*             OPTIONS COMAMID=TCP in the  */
  /*             client session before       */
  /*             using the SIGNON command.   */
  /*                                        */
  /* assumes: 1. 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 ...               */
  /*                                        */
  /* support: SAS Institute staff            */
  /*****************************************/

❶ log "NOTE: Script file
               'tcpunix.scr' entered.";

  if not tcp then goto notcp;
❷ if signoff then goto signoff;

  /*****************************************/
  /*  TCP/IP SIGNON                         */
  /*****************************************/

❸ 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)           */
  /*****************************************/
```

```
❹ input 'Userid?';
   type LF;
❺ waitfor 'Password', 30 seconds : nolog;
   input nodisplay 'Password?';
   type LF;

unx_log:
   /*******************************************/
   /* Common prompt characters are $,>,%,}    */
   /*******************************************/
❻ waitfor '$', '>', '%', '}',
      '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.       */
      /****************************************/
❼ type 'sas -dmr -comamid tcp -device
      grlink -noterminal -nosyntaxcheck' LF;
❽ waitfor 'SESSION ESTABLISHED',
      90 seconds : nosas;

❾ log 'NOTE: SAS/CONNECT
      conversation established.';
   stop;

   /*******************************************/
   /*  TCP/IP SIGNOFF                         */
   /*******************************************/
❿ 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                       */
   /*****************************************/
⓫ 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;
```

❶ 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.

❷ 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.

❸ 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.

❹ The INPUT statement displays a window with the text **`Userid?`** to allow the user to enter a server log-on user ID. The TYPE statement sends a line feed to the server to enter the user ID to the server.

❺ 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 the password is not displayed on the screen as it is entered.

❻ 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.

❼ 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.

❽ 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.

❾ 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.

❿ 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 typing **`LOGOUT`** in order to log off the server. The script then issues a LOG statement to notify the user that the connection is terminated and stops script processing.

⓫ 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 signon.

Debugging a Script

When writing 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.

CHAPTER 7

Syntax for the SIGNON and the SIGNOFF Statements and Commands

SIGNON Statement and Command

Initiates a connection between a client session and a server session.

Valid in: Client Session

Syntax

SIGNON <*options*>

Options

CONNECTREMOTE=*server-ID*
server-ID

specifies the name of the server session that you want to sign on to. Omitting this option causes the program statements to use the most recently accessed server session. You can find out which server session is current by examining the value that is assigned to the CONNECTREMOTE system option.

Alias: CREMOTE=, PROCESS=, REMOTE=

You can specify *server-ID* using different formats:

❶ *process-name*
process-name is a descriptive name that you assign to the server session on a multi-processor machine when the SASCMD= option is used.

❷ *machine-name*
machine-name is the name of a machine that is running a Telnet daemon or that is running a spawner that is not specified as a service. If the machine name is longer than 8 characters, a SAS macro variable name should be used.

❸ *machine-name.port-name*
machine-name is the name of a server, and *port-name* is the name of the port that the spawner service runs on. If the machine name is longer than 8 characters, a SAS macro variable name should be used.

❹ *machine-name.port-number*
machine-name is the name of a server, and *port-number* is the port that the spawner service runs on.

CAUTION:

Specifying *machine-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 *machine-with-port*.
- □ when used in a LIBNAME statement.

  Instead, use a one-level name or a two-level name, such as *machine-name._ _port-number*.

△

❺ *machine-with-port*
machine-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.

❻ *machine-name._ _port-number*
machine-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.

These examples of specifying *server-ID* correspond to the preceding formats.

```
1 signon emp1 sascmd="!sascmd";
2 %let sashost=hrmachine1.dorg.com; signon sashost;
3 %let sashost=hrmachine1.dorg.com; signon sashost.sasport;
4 signon hrmach1.2267;
5 %let sashost=hrmach1.dorg.com 2667; signon sashost;
6 signon hrmach1._ _2267;
```

CONNECTWAIT=YES | NO

specifies whether remote-submitted statements (WAIT= is not specified) are to be executed synchronously or asynchronously. Synchronous processing indicates that server processing must be completed before control is returned to the client session.

Asynchronous processing permits processes from multiple server sessions to execute in parallel. Control is returned to the client session immediately after an RSUBMIT begins execution to allow continued processing in the client session and in other server sessions.

If the CONNECTWAIT= option in SIGNON is omitted, the value for the CONNECTWAIT= option is resolved as follows:

1. If a value for the CONNECTWAIT= option has been specified in the RSUBMIT statement, the value for the CONNECTWAIT= option is used.
2. If the CONNECTWAIT global system option is set, the value for the global option is used.
3. If the CONNECTWAIT= option has not been specified in the SIGNON statement or as a system option, the default is to execute synchronously.

Valid values for the CONNECTWAIT= option follow:

YES | Y	specifies that RSUBMITs execute synchronously.
NO | N	specifies that RSUBMITs execute asynchronously.

If CONNECTWAIT=NO is specified, you might also specify the CMACVAR= option. Setting CMACVAR= enables you to programmatically test the status of the current asynchronous SIGNON to find out whether it has completed or is still in progress.

When %SYSRPUT executes within a synchronous (CONNECTWAIT=YES) SIGNON, the macro variable is defined to the client session as soon as it executes.

When %SYSRPUT is executed within an asynchronous (CONNECTWAIT=NO) SIGNON, the macro variable is defined in the client session when a synchronization point is encountered. To override this behavior, use the CSYSRPUTSYNC= option. For more information, see "SYSRPUTSYNC System Option" on page 24. For information about synchronization points, see "Synchronization Points" on page 146.

Note: If CONNECTWAIT=NO is specified and an autosignon is performed, an automatic SIGNOFF will not occur unless PERSIST=NO is also specified. △

Alias: CWAIT=, WAIT=

Default: YES

CMACVAR=*value*

specifies the name of the macro variable to associate with the server session.

Note: If the SIGNON command or statement fails because of incorrect syntax, the macro variable is not set. △

Except for the syntax failure condition, the macro variable (*value*) is set at the completion of SIGNON.

Valid values for CMACVAR follow:

0	the signon was successful.
1	the signon failed.
2	you have already signed on to the current server session.
3	the signon is in progress.

After signon has completed, the macro variable is set, and it becomes the default macro variable for the current server session. This default can be overridden only by specifying the CMACVAR= option in the RSUBMIT statement or command.

Alias: MACVAR=

CONNECTSTATUS=YES|NO

Values for this option follow:

YES|Y	The Transfer Status window is displayed for file transfers within the current server session. For more information, see "Transfer Status Window" on page 219.
NO|N	The Transfer Status window is not displayed for file transfers within the current server session.

If the CONNECTSTATUS= option in SIGNON is omitted, the value for the global CONNECTSTATUS= option is used. However, you can override the value for a subsequent transfer operation by using the CONNECTSTATUS= option in the RSUBMIT, PROC UPLOAD, and PROC DOWNLOAD statements.

Alias: CSTATUS=, STATUS=

Default: YES for synchronous RSUBMITs

Default: NO for asynchronous RSUBMITs

CSCRIPT=*fileref* |'*filespec*'

specifies the script file to be used during signon. Valid values are a fileref or a fully-qualified pathname that is enclosed in quotation marks. If multiple CSCRIPT= options are specified, the last specification takes precedence.

When the SIGNON command executes, the usual SAS log messages for the server session display in the LOG window of the client session. After the connection is established, the following message is displayed:

```
NOTE: Remote signon TO server-ID complete.
```

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.

'filespec'
is the name of the 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 SIGNON command. The filespec can be either a fully-qualified filename, the name of a file in the current working directory, or the location that is specified in the SASSCRIPT global option.
Do not specify a fileref and a filespec.

Alias: SCRIPT=

NOCSCRIPT
specifies that no script should be used for signon. NOCSCRIPT is useful if SASCMD= has been specified in a spawner invocation. NOCSCRIPT accelerates signon and conserves memory resources.

Alias: NOSCRIPT

USERNAME=*username* | _PROMPT_
specifies the user ID to be used when connecting to a server session. Valid values that can be assigned to USERNAME are:

username
For details about a valid user name, see "User Name and Password Naming Conventions" on page 73.

PROMPT
specifies that SAS prompt the user for a valid user name. This value enforces security.

Alias: USER=, USERID=, UID=

PASSWORD=*password* |"*encoded-password*" | _PROMPT_
specifies the password to be used when connecting to a server. The operating environment that the server runs can also affect password naming conventions. For details about password naming conventions that are imposed by the operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.
Valid values for PASSWORD are:

password
must be a valid SAS name that is 1 to 8 characters in length. 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.

"*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 PROC PWENCODE statement. For details, see the *Base SAS Procedures Guide*.

Example 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=

LOG=KEEP | PURGE | *filename* | *fileref*
OUTPUT=KEEP | PURGE | *filename* | *fileref*
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 specified file or fileref. A *backing store* is a SAS utility file that is written to disk in the client SASWORK directory. If used outside an asynchronous SIGNON, these options are ignored and the following message is displayed:

```
WARNING: LOG=/OUTPUT= options invalid with synchronous rsubmit.
Options will be ignored.
```

Valid values for both the LOG= and OUTPUT= options follow:

KEEP
spools log or output lines, as applicable, to the backing store or the client machine for later retrieval by using an RGET, RDISPLAY, synchronous RSUBMIT, or SIGNOFF statement. KEEP is the default.

PURGE
deletes all the log or output lines that are generated by the current server session. PURGE is used for saving disk resources. If you can anticipate a large volume of log data or output data to the backing store that you do not want to receive, and you want to preserve disk space, setting PURGE can be useful.

filename | *fileref*
specifies a file that is the destination for the log or output lines. The file is opened for output at the beginning of the asynchronous signon and is closed at the end of the asynchronous signon. If the first signon has completed, a subsequent signon to the same file, which you specify by using the LOG= or the OUTPUT= option, appends the contents of the current signon to the same file.

Note: Directing output to the same file for multiple asynchronous signons is not recommended. △

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, 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.
```

CAUTION:

Do not simultaneously use the asynchronous SIGNON and the PROC PRINTTO statement and re-direct output. Re-directing output by using a LOG= or an OUTPUT= option in the asynchronous SIGNON statement and using a locally submitted PROC PRINTTO statement can cause unpredictable results. △

If you use both the asynchronous SIGNON and the PROC PRINTTO statements, you might expect that the PROC PRINTTO statement forces data from the server session to be written to the file that is specified in the PROC PRINTTO statement. If that happens, 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.

However, because the asynchronous SIGNON 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.

Default: KEEP

SASCMD="*SAS-command*" | "!sascmd" | "!sascmdv" | "*host-command-file*"

is used to invoke the server session on the same multi-processor (SMP) machine. This option is most useful when using SMP hardware.

"*SAS command*"

For OpenVMS Alpha, UNIX, and Windows, specifies the command that is used to invoke the server session.

For z/OS, specifies a colon that is followed by any SAS invocation options.

Example:

```
sascmd=":ls=256"
```

"!sascmd"

For OpenVMS Alpha, UNIX, and Windows, starts a server session by using the same command that was used to invoke the client session

"!sascmdv"

For OpenVMS Alpha, UNIX, and Windows, starts a server session by using the same command that was used to invoke the client session and writes the SAS invocation to the log.

SASCMD= is also a global system option. If the SASCMD= global option is already set, the SASCMD= option that is specified in the SIGNON statement or command takes precedence over the global system option. For more information, see "SASCMD= System Option" on page 20.

An example of a typical setting for the SASCMD= option follows:

```
signon sascmd="sas"
```

As another example, commands that contain spaces must be enclosed in double quotation marks.

```
signon rmthost sascmd='"c:\Program Files\SAS\SAS System\9.1\sas.exe"';
```

In order to execute additional commands prior to SAS invocation, you might write a command file that is specific to your operating environment. File-naming conventions conform to the requirements of the operating environment, as follows.

Table 7.1 Filename Extensions by Operating Environment

Operating Environments	**Filename Extensions**
OpenVMS Alpha	.com
UNIX	.ksh*
Windows	.bat, .cmd

* The UNIX filename extension depends on the specific shell being used; for example, Korn shell, C shell, or Bourne shell.

Note: The SASCMD= option does not support z/OS command files. △

The TCP/IP access method adds options, such as -DMR, to the server's SAS command.

For Windows, the TCP/IP access method also appends the following options:

- -COMAMID TCP
- -ICON
- -NOLOGO
- -NOTERMINAL

An example of creating a command file named *mysas.bat* for Windows follows:

```
cd "C:\Program Files\alpair\SAS\V9.1"
mkdir mywork
sas %* -nosyntaxcheck -work "mywork"
```

*%** adds the appended TCP/IP access method options to the SAS invocation in *mysas.bat*.

To execute the command file, specify its name as the value for SASCMD=. For example:

```
signon smp1 sascmd="mysas.bat";
```

CAUTION:

OpenVMS Alpha operating environment only: If the NODETACH system option is set, and if multiple server sessions are running under OpenVMS Alpha and you observe degraded performance, you will be alerted to the condition with an error message. For example:

```
ERROR: Process quota exceeded.
ERROR: Cannot spawn subprocess. Check process limit quotas and privileges.
```

NODETACH causes a signon to occur in a subprocess of the parent's process, which can deplete resources. If NODETACH is set, try setting the DETACH system option, which causes signons to occur as detached processes rather than as subprocesses. For more information about NODETACH, see the *SAS Companion for OpenVMS Alpha*.

When using NODETACH to improve performance, ask your system administrator to set the following resources to the specified values per server signon:

Table 7.2 OpenVMS Operating Environment Resource Values

User Account Resource	Minimum Value
Paging file quota	40000
Buffered I/O byte count quota	13000
Open file quota	65
Subprocess limit	1
Timer queue entry limit	1 to 8

Signon via the SASCMD= option is not supported when running SAS from a captive OpenVMS account. SASCMD signons require the process in which SAS is running to start a remote SAS session, either in a subprocess or in a detached process. Starting subprocesses is not allowed under a captive account. A detached process that runs under a captive account will not be able to invoke SAS because a captive OpenVMS account is under the control of the login command procedure. The command language interpreter will execute only the commands in your login command procedure and then the process will exit.

The **!sascmdv** value in the SASCMD= option causes the display of a symbol that specifies how the server session was started. You can print the symbol's value by using the **getsym** DATA step function.

Example:

```
rsubmit;
   %put %bquote(
   %sysfunc (getsym(SASCMD_2042CF6B)));
endrsubmit;
```

△

NOTIFY=YES|NO

requests the display of a notification message window to report the completion of an asynchronous RSUBMIT. The message includes the server session ID, which is TASK1, in the following example.

```
Asynchronous task TASK1 has completed
```

The display window does not interfere with any other SAS session executions in progress. To acknowledge the message and to close the window, click OK.

Valid values follow:

YES|Y	enables notification.
NO|N	disables notification. This is the default.

This option is applied only during signon and autosignon. After being set, notification remains in effect for the duration of the server session.

A notification message window is displayed:

- upon completion of an asynchronous signon.
- for all SAS session execution modes (DMS, line, and batch) that a terminal is attached to. You associate a terminal with a SAS session by setting the TERMINAL option.

  If you try to set the NOTIFY= option in a SAS session that has been invoked with the NOTERMINAL option, the following message is displayed:

  ```
  WARNING: The NOTIFY option is valid only if a TERMINAL is attached to this
  SAS session. Option will be ignored.
  ```

To disable notification during a session, you must sign off the server session and then sign on to the server session again, and either omit the NOTIFY= option or specify NOTIFY=NO in the SIGNON statement.

If notification is set and you issue statements or commands (such as RGET or SIGNOFF) during the asynchronous signon that changes processing from asynchronous to synchronous mode, the notification window is *not* displayed.

Default: NO

SERVER="*Connect-server-definition*"
specifies a CONNECT server definition that has been defined in a SAS Metadata Repository. SAS Management Console (SMC) can be used to create, update, and delete server definitions.

Note: Because the CONNECT server definition configures *all* SAS/CONNECT server options, you do not need to specify any other options in the SIGNON= statement. △

If you have not set global options to specify how to connect to the SAS Metadata Server, a requestor window will be displayed in which you can configure the IP address for the server, the port, the protocol, and the user name and password.

Note: The CONNECT server definition must be enclosed in quotation marks. △

For complete details about creating and populating a SAS Metadata Repository, see the documentation at **`support.sas.com/rnd/eai/openmeta`**.

SERVERV="*Connect-server-definition*" | _ALL_
is used to view the values that are configured in the CONNECT Server definition.

"*Connect-server-definition*"
displays the values for the specified CONNECT server definition that is stored in the SAS Metadata Repository.

ALL
displays the values for all CONNECT server definitions that are stored in the SAS Metadata Repository.

The following example displays the values that are specified for the CONNECT server definition that is named hrmach1.

```
Server=                 hrmach1 --- SAS/CONNECT Server
Remote Session ID=      sashost
ServerComponentID=      A5Z3NRQF.AR00005L
Remote Host=            hrmach1.dorg.com
Communication Protocol= TCP
Port=                   2267
Scriptpath=             tcpunix.scr
Tbufsize=               4096
Macvar=                 hrmach_macvar
Inheritlib=             (work=cwork)
Wait=                   No
SignonWait=             No
Status=                 No
Notify=                 Yes
Netencrypt=             No
```

SIGNONWAIT=YES | NO
SIGNONWAIT specifies whether the signon is executed synchronously or asynchronously.

YES | Y	specifies that SIGNON will execute synchronously. Synchronous processing means that a signon to a server session must be completed before control is returned to the client session.
NO | N	specifies that a SIGNON will execute asynchronously. Asynchronous processing permits signons to multiple server sessions to execute in parallel. Control is returned to the client session immediately after the SIGNON statement is issued.

You can use SIGNONWAIT=NO 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 remote submitting units of work to a server session, as soon as signon is complete.

Note: If SIGNONWAIT=NO, the USERID= and PASSWORD= options cannot be set to the value of _PROMPT_. △

You can use these methods to find out when signon has completed:

- □ LISTTASK statement in the RSUBMIT statement (for details, see "LISTTASK Statement" on page 151)
- □ CMACVAR= option in the SIGNON statement.

SIGNONWAIT is also a global option. If the SIGNONWAIT global option is set, the SIGNONWAIT= option that is specified, locally, in the SIGNON statement or command takes precedence over the global system option. If SIGONWAIT *is* set as a global option and SIGNONWAIT= is *not* set as an option in SIGNON, the global option setting will apply to the SIGNON statement. For details, see "SIGNONWAIT System Option" on page 23.

Default: YES

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 SASWORK library cannot be reassigned in any SAS session, you cannot reassign it in the server session either. △

Example:

```
libname scorcard '.';
rsubmit inheritlib=(scorcard work=cwork);
  libname scorcard list;
  libname cwork list;
  data scorcard.a; x=1; run;
endrsubmit;
proc datasets lib=scorcard; run;
```

TBUFSIZE=*buffer-size-in-bytes*

specifies the size of the buffer that SAS/CONNECT uses for transferring data between a client and a server across a network.

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.

TBUFSIZE= is also a global option.

- □ If TBUFSIZE= is specified as an option in the SIGNON statement, it takes precedence over the TBUFSIZE= global option.

- If TBUFSIZE= is specified as a global option at the client and the server, the value at the client takes precedence.
- If TBUFSIZE= is specified as a global option at the client but is *not* specified in the SIGNON statement, the global option value will be used.
- If the TBUFSIZE= global option is specified at the server but *not* at the client, the value at the server will be used.
- If TBUFSIZE= is *not* set as a global option or as an option in SIGNON, the default of 32768 will be used.

Default: 32768 bytes

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 signon is processed either synchronously or asynchronously.

synchronous
: Client session control is not regained until after the signon 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 signon is in progress.

Synchronous signons 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 signon entails accessing the machine that the server session will run on and then invoking a SAS/CONNECT server session.

An autosignon is an implicit signon to the server when the client issues a remote submit request for server processing. When the AUTOSIGNON global system option is set, the RSUBMIT command or statement automatically executes a SIGNON and uses any SAS/CONNECT system global options in addition to any connection options that are specified with RSUBMIT. For example, if you specify either the NOCONNECTWAIT global system option or the CONNECTWAIT=NO option in the RSUBMIT command or statement, asynchronous RSUBMITs will be the default for the entire connection.

User Name and Password Naming Conventions

Each user name and password is limited to 256 characters that follow these conventions:

- Mixed case is allowed.
- A null value, which is no value, that is delimited with quotation marks is allowed.

- □ Quotation marks must enclose values that contain one or more spaces.
- □ Quotation marks must enclose values that contain one or more special characters.
- □ Quotation marks must enclose values that contain one or more quotation marks.
- □ Quotation marks must enclose values that begin with a numeric value.
- □ Quotation marks must enclose values that do not conform to rules for user-supplied SAS names. For details about rules, see *SAS Language Reference: Dictionary*.

Examples:

```
user=joe password=Born2run;
user=joe password='' # null space specified by contiguous quotation marks;
user='joe black' password='Born 2 run';
user='joe?black' password='Born 2 run';
user='apexdomain\joe' password='2bornot2b' # Win NT user name;
user='"happy joe"' pw=_prompt_;
user=_prompt_;
userid="myuserid" password="{sas001}c2Vydm1hY2g";
```

Examples

Example 1: Signon by Using a Script The OPTIONS statement specifies the server-ID, and the FILENAME statement identifies the sign-on script. The SIGNON statement initiates the connection. The TCP/IP access method is assumed by default.

```
options remote=rhost;
filename rlink 'external-file-name';
signon;
```

Example 2: Secured Signon by Using an Encoded Password The USERNAME= and PASSWORD=options in a SIGNON statement ensure a secured signon. At signon, the user is prompted for a user name and password, which is automatically supplied in its encoded form. For details, see the PASSWORD= option in the SIGNON statement on page 66.

```
signon user=_prompt_ password="{sas001}MVNoYXJl";
```

Example 3: Creating a Sign-on 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 script named *mysas.bat* for Windows:

```
set userdrive=%1
%userdrive%
mkdir \sassdir
cd \sassdir
"C:\Program Files\SAS\SAS 9.1\sas" -nosyntaxcheck
-work "mywork" %2 %3 %4 %5 %6 %7 %8 %9
```

To execute the command file, specify its name as the value for SASCMD=.

```
signon sascmd="mysas.bat sysjobid";
```

Example 4: Signing 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 WNT runs an unsecured spawner under Windows NT.
- 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:

```
❶ options comamid=tcp;
signon wnt;

   /*****************************************/
   /* initiates connection to an OS/390 server host*/
   /*****************************************/
❷ filename tsoscr '!sasroot/misc/connect/tcptso.scr';
signon tso script=tsoscr;

❸  /******************************************/
     /* submit statements to a Windows NT server    */
     /*****************************************/
rsubmit wnt wait=no;
   statements to be processed by Windows NT server

endrsubmit;

❹  /******************************************/
   /* submit statements to z/OS server */
   /*****************************************/
rsubmit tso wait=no;
   statements to be processed by z/OS server
endrsubmit;
❺ waitfor _ALL_ wnt tso;
   /*****************************************/
   /* ends both connections                  */
   /*****************************************/
❻signoff tso cscript=tsoscr;
signoff wnt cscript=winscr;
```

❶ The client signs on to the server session WNT.
❷ The client uses a script to sign on to the server session TSO.
❸ The WNT server session asynchronously processes the statements that are enclosed by the RSUBMIT and ENDRSUBMIT statements.
❹ The TSO server session asynchronously processes the statements that are enclosed by the RSUBMIT and ENDRSUBMIT statements.
❺ The client session waits for both RSUBMITs to complete.
❻ The client uses scripts to sign off both server sessions.

Example 5: Using MACVAR to Test for a Successful Signon The following example illustrates that the macro variable from a successful signon 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 Command and Statement

Ends the connection between a client session and a server session.

Valid in: Client session

Syntax

SIGNOFF <*options*> ;

Options

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, *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=

CMACVAR=*value*

specifies the name of the macro variable to associate with the signoff.

Note: If the SIGNOFF command fails because of incorrect syntax, the macro variable is not set. △

Except for this condition, the macro variable is set after the SIGNOFF command is completed.

Valid values for CMACVAR= follow:

0	indicates that the SIGNOFF was successful.
1	indicates that the SIGNOFF failed.

If the SIGNOFF succeeds, the macro variable is set to 0. However, because the connection ends, the macro variable is no longer associated with that server session.

If the SIGNOFF fails, the macro variable is set to 1. Although the signoff is unsuccessful, the macro variable is no longer associated with that server session. Instead the default MACVAR= (if any) is used.

Alias: MACVAR=

CSCRIPT=*value*

specifies the script file to be used during signoff. 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 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 script should be used for signoff. NOCSCRIPT is useful if you have defined the RLINK fileref but do not want to use it during signoff. NOCSCRIPT accelerates signoff 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.

Examples

Example 1: Setting a Macro Variable at Signon Checks for Signoff Failure In this example, a macro variable is assigned at signon. Therefore, if the signoff fails, the macro variable will be set for this server session.

```
/* Signon successful, rhost1 will be  */
/* set to 0 to indicate success, and  */
```

```
     /* macro variable rhost1 is now       */
     /* associated with this server        */
     /* session.                           */
  signon rhost macvar=rhost1;

     /* Signoff 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 macvar=rhost2
     script='noexist.scr';
```

Example 2: Not Setting a Macro Variable at Signon Does Not Check for Signoff Failure In this example, a macro variable is not assigned at signon. Therefore, if the signoff fails, the macro variable will not be set for this server session.

```
     /* No macro variable associated with  */
     /* server session                     */
  signon rhost;

     /* Signoff will fail, and ABC will    */
     /* be set to 1 to indicate this,      */
     /* but because it was unsuccessful,   */
     /* the default of no macro variable   */
     /* will go into effect for this       */
     /* server session.                    */
  signoff rhost macvar=abc
     cscript='noexist.scr';
```

When the SIGNOFF command executes, the usual SAS log messages for the server session appear in the Log window of the client session. After the connection ends, the following message is displayed:

```
NOTE: Remote signoff to server-ID complete.
```

Example 3: Simple Signoff for a Single Session The following FILENAME statement assigns the fileref RLINK to a 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 4: Signoff from a Specific Session If multiple server sessions are executing, you can specify which *server-ID* to sign off from.

```
signoff ahost;
```

Example 5: Signoff from Session Using Specific Script Fileref If you assign another fileref, which is not the default, to the script:

```
filename endit 'external-file-name';
```

you must specify the fileref in the SIGNOFF statement because it is not the default script fileref.

```
signoff cscript=endit;
```

Example 6: Signoff by Using a Nondefault Fileref When Multiple Sessions Are Running If you use RLINK or any other fileref in the SIGNOFF command or statement, you can define the fileref for the script in a FILENAME statement in the SAS autoexec file. Automating the FILENAME statement in the autoexec file provides a convenience at signoff.

If you do not assign a fileref to the script, you must specify the filespec in the SIGNOFF command.

```
signoff cscript='external-file-name';
```

Example 7: Signoff 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;
```

FILENAME Statement

Associates a SAS fileref with an external file.

Valid in: Client and server sessions

Syntax

FILENAME *fileref 'filespec'* <*host-options*>;

Options

fileref
specifies any SAS file reference name.

'filespec'
specifies the physical name of an external file so that the external file is recognized by the operating environment.

host-options
specify details, such as file attributes and processing attributes that are specific to the operating environment.

Details

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.

Using a FILENAME Statement for Script Files

A common use of the FILENAME statement is to define filerefs for script files. A script's fileref can then be specified in SIGNON and SIGNOFF commands to identify the script that starts or ends the connection.

You can define a default fileref for a script 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.

Using 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 type 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*.

Using 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: Using a FILENAME Statement for a Script File Suppose your SAS support consultant writes a script and copies it to a directory in your client environment. To define the default fileref RLINK for your script, use the following FILENAME statement:

```
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:

```
signon;
```

To end the connection, enter:

```
signoff;
```

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:

```
signoff endit;
```

Example 2: Using 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 remote submit the following statements to assign the fileref in the server session and to perform the download:

```
filename rhost 'server-file-name';

   proc download infile=rhost outfile=lhost;
   run;
```

For more examples of using the FILENAME statement and the DOWNLOAD and UPLOAD procedures, see Chapter 23, “Using Data Transfer Services,” on page 215.

CHAPTER 8 SAS Component Language (SCL) Functions and Options

COMAMID SCL Function

Returns a string that contains all of the COMAMIDs 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 (COMAMIDs) 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 COMAMIDs 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 COMAMIDs 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 listbox. Although this example does not include it, you would specify that the listbox 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
the session identifier (REMOTE=*server-ID*)

characters 18 through 57
the description

characters 58 through 61
is 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 non-zero 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 8 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);
```

Using 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). When you set the SASSCRIPT= system option, it generates another SAS system option, SASFRSCR, which 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;
```

CHAPTER 9 Script Statements

Summary of Script Statements

Table 9.1 Summary of Script Statements

Statement	Purpose
ABORT	stops execution of a script immediately and signals an error condition
CALL	invokes a routine
ECHO	controls the display of characters that are sent from the server session while a WAITFOR statement executes
GOTO	re-directs execution to the specified script statement
IF	checks conditions before the execution of labeled script statements
INPUT	displays a prompt to the user that requests a response for the server session
LOG	sends a message to the client session SAS LOG window
NOTIFY	sends a message in a window to the client session
RETURN	signals the end of a routine
SCANFOR	specifies a pause until conditions are met; is an alias for WAITFOR
STOP	stops execution of a script under normal conditions
TRACE	displays script statements as they execute
TYPE	sends characters to the server session as if they were typed at a terminal
WAITFOR	specifies a pause until conditions are met

ABORT

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.

CALL

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

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

Re-directs 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

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 if 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 signon or signoff, *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. For more information about COMAMID= values for emulation software, see "COMAMID= System Option" on page 14.

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

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

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

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

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

Specifies a pause until conditions are met. Is an alias for WAITFOR.

Syntax

SCANFOR *pause-specification-1* <*... pause-specification-n*>;

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

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

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

Sends characters to the server as if they were typed at a terminal.

Syntax

TYPE *text*;

Syntax Description

text
is the user-specified string of characters sent to the server.

Details

The TYPE statement sends characters to the server as if they had been typed on a terminal 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'.
- hex 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 typed, 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, to divide the following TYPE statement:

```
type "sas options ('dmr comamid=tcp')"
enter;
```

change it to:

```
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 9.2 ASCII Character Mnemonics

ASCII Control Character	Mnemonic Representation	Decimal Value	Hexadecimal Value
Line feed	LF or CTL_J	10	0A
Carriage return	CR or CTL_M	13	0D

WAITFOR

Specifies a pause until specific conditions are met.

Syntax

WAITFOR *pause-specification-1*<. . . *pause-specification-n* >;

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>

where

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

- □ a character literal that is enclosed in quotation marks
- □ a hex 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 in Chapter 5, "Starting and Stopping SAS/CONNECT," on page 43, 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.

Usage Notes

- 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, 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).

CHAPTER 10 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 machine 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 in the client session to see what is displayed by the server session at the time 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 100.

Requested-Link-Not-Found Message

While signing 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 *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

For more information about recovery from this error, see "SAS/CONNECT Server Session Initialization Errors" on page 100.

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:

- an invalid option name or value might have been specified in the SAS command
- the user might not be authorized by the machine that the server session runs on to execute the SAS program modules, 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.

SAS Console Log Messages for Windows

The SAS console log 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 machine 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.

SAS Console Log Messages for 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.

SASCMD= signon
: Standard output is piped to the SAS session that issued the sign-on statement. The standard output messages are written to the SAS log in the SAS session. Each message contains a prefix that identifies the server session (the server ID) that was being created.

Spawner signon
: 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 re-directed to a specific location. An example of re-directing standard output to a log follows:

```
sastcpd -nocleartext > spawner.log
```

SAS console log messages will be directed to the standard output location. For details about the UNIX spawner, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Telnet daemon signon
: The standard output location for the SAS session is the script processor in the SAS session that issued the sign-on 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.

SAS Console Log Messages for z/OS

The SAS console log is written to the SASCLOG ddname of the SAS session that is started. The location of the SASCLOG ddname varies according to the sign-on method that was used.

SASCMD= signon
: The SASCLOG is written to the SYSOUT device.

To locate messages in the SAS console log, you must find the appropriate user ID in the spooled files. You can use a Job Entry System (JES) spool viewer (such as SDSF or EJES) to browse the spooled files.

Spawner signon
: The SASCLOG is written to the SYSOUT device.

To locate messages in the SAS console log, you must find the appropriate user ID in the spooled files. You can use a Job Entry System (JES) spool viewer (such as SDSF or EJES) to browse the spooled files.

Telnet daemon signon
: The SASCLOG ddname is directed to the script processor in the SAS session that issued the sign-on 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.

P A R T 4

Compute Services

CHAPTER 11 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 Chapter 12, "Syntax for RSUBMIT Statement and Command," on page 131.

Here are some of the benefits of Compute Services:

- Gives you access to additional CPU resources.

  You might have multi-processor SMP machines or remote machines on your network that are underutilized. These CPUs could be used to execute the CPU intensive portions of your application faster and more efficiently than your local machine. 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 machine 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. By moving the processing to the machine where the data resides, there is no 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 machines make file transfers convenient and easy, large files do not move quickly between machines. 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 machine while running them and keeping the data that they use on a different machine.

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

MP CONNECT

Prior to SAS 8, when an RSUBMIT was executed, the client session was suspended until processing by the server session had completed. In SAS 8, MP CONNECT functionality was added, which allows you to execute RSUBMITs 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 RSUBMITs to one or more additional server sessions. Asynchronous RSUBMITs are most useful for longer-running tasks.

MP CONNECT enables you to perform multi-processing with SAS by establishing a connection between multiple SAS sessions, and enabling each of the sessions to asynchronously execute tasks in parallel. You also have the ability to 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 machine has been greatly simplified. This gives you the ability to exploit SMP (Symmetric Multi-Processing) 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 *n* SAS processes, where *n* is the number of units of work that you want to perform in parallel. SAS processes that are started on a single multi-processor machine 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 it’s 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 possesses 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 Chapter 12, “Syntax for RSUBMIT Statement and Command,” on page 131.

Independent Parallelism

Overview of Independent Parallelism

Independent parallelism is possible when the execution of Task A and Task B do not have any interdependencies. For example, if an application needed 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 PROC SORTs can be performed in parallel and 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 PROC SORTs. The merge would be executed in the client session after the two parallel SORTs are completed.

Considerations for Independent Parallelism

When using MP CONNECT (especially on an SMP machine), you want to be sure 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. Additionally, 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 machine, each session has its own WORK library. Because each WORK library for each SAS process is created in the same temporary file directory by default, you have multiple SAS processes performing intensive I/O to their respective WORK libraries, but all these WORK libraries exist on the same physical disk. This is another potential I/O bottleneck, which 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. Using the SPD Engine, the data sets that are created can be partitioned 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 will be stored in the USER library.

Note: When using MP CONNECT on multiple remote machines, the WORK library of the remote sessions exists on the individual machines 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, when a SAS DATA step is 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 machines. 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 Chapter 20, "Syntax for the LIBNAME Statement, SASESOCK Engine," on page 199. For an example of piping, see "Example 6: Using MP CONNECT with Piping" on page 161.

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
- GANT 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 machine 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 Chapter 19, "Syntax for the LIBNAME Statement," on page 195.
- Ensure that the port that the output is written to is on the same machine that the asynchronous process is running on. However, a SAS procedure that is reading from that port can be running on another machine.
- Be sure that the task that reads the data does not complete before the task that writes the data. For example, if one process has 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 only printing 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 prior to 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: Preventing Pipes from Closing Prematurely" on page 162.

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 machine or across multiple machines on your network.

MP CONNECT uses hardware resources that you might have thought were out-dated and useless. Using MP CONNECT, you can put multiple, slow, inexpensive machines 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 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 would include an SMP machine or multiple machines on a network with each machine 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 machines on a network. For an SMP machine, 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 a symmetric multi-processing (SMP) machine, but they can also be run on multiple remote single or multi-processor machines on a network. When running multiple SAS processes on an SMP machine, 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 machine but cannot be executed across machines. 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 machine. *Scaling up* also means to leverage the multiple processors, disk drives, and I/O channels on a single server machine.

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 machine 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 make use of multiple processors in SMP hardware. Multi-threading has been incorporated into SAS 9 (and later), including many SAS servers, several performance-critical SAS procedures, and many SAS engines. Multi-threading 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 machine, the operating environment can schedule the processes on different processors to use all the hardware resources on the machine. In addition, by running multiple SAS processes across the machines that are available on a network, you can use idle processors and put multiple, slow, inexpensive machines to work in parallel on a job and turn them into a valuable, powerful, inexpensive computing resource.

Multi-threading 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 multi-threading. When running multiple processes by using multiple threads on an SMP machine, 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 machines with each SAS process running on a physically separate machine, it might be better to let the threading within the process fully use the individual machines.

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 machines, 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.

Monitoring 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 Command and RDISPLAY Statement" on page 144.

As an alternative to RDISPLAY, you can use SAS Explorer's Monitor window. For details, see "Using SAS Explorer to Monitor SAS/CONNECT Tasks" on page 114.

Managing 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 Command and RDISPLAY Statement" on page 144

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 re-directing to a file by using the LOG= option and the OUTPUT= option. For details about RGET, see "RGET Command and RGET Statement "on page 144. For details about the LOG= option and the OUTPUT= option, see "RSUBMIT Statement and Command" on page 131.

MP CONNECT Task Completion

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

- LISTTASK statement
- CONNECT Monitor window from the SAS Explorer
- 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 Statement" on page 151

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 "Using SAS Explorer to Monitor SAS/CONNECT Tasks" on page 114.

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 Statement and Command" on page 131.

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 Statement and Command" on page 131.

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 Statement" on page 150.

Using 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 machine that is equipped with SMP hardware, which facilitates multi-processing.

To start the SAS/CONNECT Monitor window, from the pull-down menu, select:

View ► SAS/CONNECT Monitor

The SAS/CONNECT Monitor window 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 window, and selecting the Kill option from the menu that displays 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. For details about ODS, see the *SAS Output Delivery System: User's Guide*.

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

- □ Remote 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.
- □ 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.
- □ Remote 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.

For all scenarios that use asynchronous processing, use the RGET statement or command to retrieve results. The output is not available until the results are retrieved. (For more information, see "RGET Command and RGET Statement "on page 144). The accumulated output is retrieved and transferred to the client session.

Using the SAS Windowing Environment to Control Remote Processing

Overview of Remote Processing Control Using the SAS Windowing Environment

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

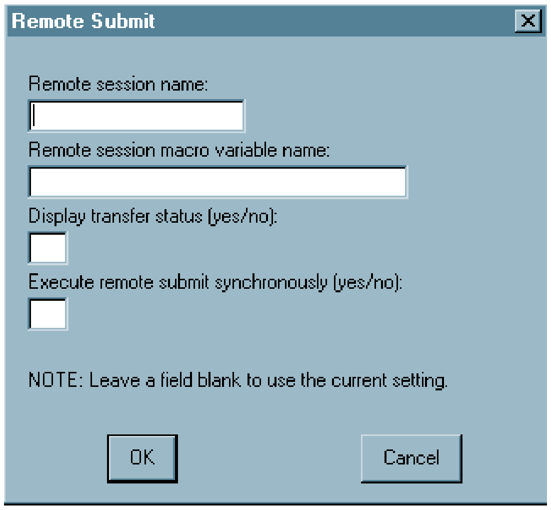
Remote Submit

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

Run ► Remote Submit

The Remote Submit dialog box appears.

Display 11.1 Remote Submit Dialog Box


Remote Submit

Remote session name:

Remote session macro variable name:

Display transfer status (yes/no):

Execute remote submit synchronously (yes/no):

NOTE: Leave a field blank to use the current setting.

Field explanations follow:

`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 Statement and Command" on page 131.

`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 Statement and Command" on page 131.

`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 Statement and Command" on page 131.

`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 Statement and Command" on page 131.

CAUTION:

Remote Submit Limitation: The Remote Submit pull-down 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 pull-down 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 from the menu bar, select

Run ► Remote Get

Field explanations follow:

`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 Command and RGET Statement "on page 144.

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 from the menu bar, select

Run ▶ Remote Display

Field explanations follow:

`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 Command and RDISPLAY Statement" on page 144.

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`**. △

Interaction between Compute Services and Macro Processing

Macro-Generated RSUBMIT Blocks

Macros are compiled into macro program statements by the macro processor. A macro-generated RSUBMIT refers to an RSUBMIT/ENDRSUBMIT statement block that is contained within a macro definition. The general structure of this block follows:

```
/* begin container macro */
%MACRO macro-name;
RSUBMIT;
  statements
ENDRSUBMIT;
%mend macro-name;
/* end container macro */
%macro-name
```

Macro processing within a macro-generated RSUBMIT might not always produce the results that you expect. Here are the types of statements that can be included in a macro-generated RSUBMIT/ENDRSUBMIT statement block:

- macro definition
- SAS statements that are not macro statements or macro definitions
- macro statements.

Only the first and second statement types are always executed in the server session. Macro statements can be resolved and executed in the client session rather than in the server session.

Macro Definitions

When the macro processor encounters a macro definition in a macro-generated RSUBMIT statement, all the statements that follow the %MACRO statement are compiled into macro program statements until a %MEND statement is compiled. Then, the embedded macro definition is remote-submitted to the server session, and the macro is defined in the server session when the "container" macro is invoked.

SAS Statements That Are Not Macros or Macro Definitions

When the macro processor encounters statements that are not macro definitions or macro statements, such as SAS procedure statements or DATA steps, in a macro-generated RSUBMIT statement, these statements are compiled into macro program statements. When the macro is executed, these statements are remote-submitted to the server session for execution.

Macro Statements

Macro statements that you include in a macro-generated RSUBMIT statement might get resolved and executed in the client session rather than in the server session, regardless of your including the statements in an RSUBMIT/ENDRSUBMIT statement block. The macro processor in the client session resolves variables that are specified in the following statements:

%DO

%IF

%LET

%PUT

%SYSRPUT

The macro processor also compiles the variables into macro program statements, which the container macro executes in the client session. If you do *not* want the statements to execute in the client session, you can use various programming techniques to control the location where the statements execute.

Ensuring That the RSUBMIT Statements Are Executed in the Correct Session

Programming Techniques

To avoid possible macro processing confusion, you can use specific programming techniques to ensure that macro statements are processed in the server session or in the client session, whichever you choose.

%SYSLPUT Statement

To assign a value to a macro variable in a server session, use the %SYSLPUT macro statement. Using the %SYSLPUT statement to define a macro variable and then using a macro variable in the server session is better than attempting to remote-submit a %LET macro statement. The syntax of the %SYSLPUT statement follows:

%SYSLPUT *macvar* = *value*</REMOTE=*session-ID*>;

Example:

```
%syslput remvar1=%sysfunc(date(),date9.);
```

The client session evaluates the value that is assigned to the server session macro variable REMVAR1. If the macro variable REMVAR1 does not exist, it is created. Using %SYSLPUT prevents the macro processor from interpreting a %LET statement that is in the macro-generated RSUBMIT statement in the client session.

%NRSTR Macro Quoting Function

If a special character or a mnemonic affects the way the macro processor constructs macro program statements, use the %NRSTR macro quoting function to mask the item during macro compilation (or during the compilation of a macro program statement in open code). %NRSTR can be used to mask the macro statements, which causes the macro processor to ignore the macro program statements in the client session and forces the macro statements to be executed in the server session.

The syntax for the %NRSTR quoting function when used with a macro statement follows:

```
%NRSTR (%%) macro-statement;
```

Example:

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

%NRSTR prevents the macro processor from interpreting a macro statement that is in the RSUBMIT statement in the client session. %NRSTR causes the macro statement to be interpreted and executed in the server session. For details about macros, see *SAS Macro Language: Reference*.

Examples

Client Session Execution: Macro Statement in RSUBMIT

```
/* In this macro, %LET is a macro statement that will be interpreted  */
/* by the client session and not remote-submitted.     */
/* If REMVAR1 is not already defined in the server session,               */
/* this example will produce an error.     */
%macro example;
%global remvar1;

rsubmit;
  data x; x=1; run;
  %let remvar1=%sysfunc(date(),date9.);
  data a; x="&remvar1"; run;
endrsubmit;

%mend;
%example;
```

Server Session Execution: %SYSLPUT to Mask Client Session Macro Processing

```
/* In this macro, the %SYSLPUT statement is used to assign a value to a         */
/* macro variable in the server session, to avoid having the client session */
/* macro processor interpret a %LET statement in the RSUBMIT block.         */
/* %SYSLPUT can also be issued outside the macro definition.                */

%macro example1;

%syslput remvar1=&sysfunc(date(),date9.);
rsubmit;
  data a; x="&remvar1"; run;
endrsubmit;

%mend;
%example1;
```

Server Session Execution: %NRSTR to Mask Client Session Macro Processing

```
/* In this macro, %NRSTR is used with the %LET macro statement to "hide"           */
/* it from the client session macro processor and allow it to be remote submitted.  */

%macro example2;

rsubmit;
   %nrstr(%%)let remvar1=%sysfunc(date(),date9.);
   data a; x="&remvar1"; run;
endrsubmit;
%mend;
%example2;
```

Server Session Execution: Macro Definition in an RSUBMIT Block

```
/* This shows a macro definition embedded in an RSUBMIT block. The entire */
/* ONREMOTE macro definition is remote submitted and none of the          */
/* statements in the ONREMOTE macro are interpreted by the macro          */
/* processor in the client session.                                       */

%macro example3;

rsubmit;
  %macro onremote;
  %global abc;
     %put this is on the server;
     %let abc=value;
  %mend;
  %onremote;
endrsubmit;
%mend;
%example3;
```

Local Execution: %IF Allows Conditional Processing Based on Client Macro Variable

```
/* In this macro example, %IF is interpreted by the              */
/* macro processor in the client session in order to determine   */
/* whether to execute PROC DOWNLOAD.                             */
%macro example4;
%global localvar2;
rsubmit;
  data remds;
    x=1;
  run;
  %if &localvar2 eq getit %then %do;
     proc download;
       run;
  %end;
endrsubmit;
%mend;
%let localvar2=getit;
%example4;                    /* download occurs */
%let localvar2=;
%example4;                    /* download does not occur */
```

Client and Server Session Execution: %PUT Statement Defined in Nested Macros

```
/* The following macro shows how embedded macros work.  The         */
/* %PUT statements indicate where the macros are defined and        */
/* where they should be invoked.                                    */
/* The macro ONREMOTE is defined to the server session because it   */
/* is in an RSUBMIT/ENDRSUBMIT block.  Therefore, its invocation    */
/* must be remote submitted.  The macro ONLOCAL is defined to       */
/* the client session and its invocation is locally submitted.      */
%macro embeddedmacros;
rsubmit;
```

```
  %macro onremote;
    %put on the remote side;
  %mend;
endrsubmit;
%macro onlocal;
  %put on the local side;
%mend;
rsubmit;
  %onremote;
endrsubmit;
%onlocal;
%mend;

%embeddedmacros;
```

Server Session Execution: No Macros or Macro Statements in Macro-Generated RSUBMIT

```
/* This macro shows that everything in the RSUBMIT/ENDRSUBMIT block     */
/* is executed by the server session because there are no macro         */
/* statements in the macro generated RSUBMIT to be interpreted by       */
/* the macro processor in the client session.                           */

%macro do-x;
rsubmit;

data x;
  date="04 July 03";
  put date=;
run;
endrsubmit;
%mend;
%do-x;
```

Server Session Execution: %NRSTR to Mask Local Macro Processing

```
/* This macro uses SYMPUT in an RSUBMIT, and                           */
/* uses %NRSTR to "hide" the %PUT statement from the macro processor   */
/* in the client session, so that it can be executed by the        */
/* server session.                                                     */
%macro nullds;
rsubmit;
  data _null_;
    call symput('abc','abc');
    call symput('one','1');
    call symput('date',"%sysfunc(date(),date9.)");
  run;
%nrstr(%%)put abc=&abc one=&one date=&date;
endrsubmit;
%mend;
%nullds;
```

Frequently Asked Questions

Will %SYSFUNC Be Evaluated in the Client Session or the Server Session?

Whether %SYSFUNC is evaluated in the client or the server session depends on how %SYSFUNC is used. If it is used in a %LET or a %PUT macro statement, %SYSFUNC is executed in the client session. However, you can use %NRSTR in your macro definition to mask the %LET and %PUT statements, which causes the %LET, %PUT, and %SYSFUNC macros to be executed in the server session. In the following example, %SYSFUNC executes in the remote session because %NRSTR is used.

```
%macro remotesysfunc;
  rsubmit;
    %nrstr(%%)let current="%sysfunc(time(),time.)";
    %nrstr(%%)put current=&current;
  endrsubmit;
%mend;
%remotesysfunc;
```

In the next example, %SYSFUNC is not part of a macro statement; it is part of the DATA step. Therefore, including it in an RSUBMIT block causes it to be executed in a server session.

```
%macro dssysfunc;
rsubmit;
  data x;
    time="%sysfunc(time(),time.)";
    put time=;
  run;
endrsubmit;
%mend;
%dssysfunc;
```

Does %SYSLPUT Affect the Current Session or All Sessions?

I don't want %SYSLPUT to affect all my sessions because I am passing an ID to my server session.

%SYSLPUT affects either the server session that is specified by using the /REMOTE= option or the current server session. The current session is the one that you have most recently accessed. You can find out which server session is current by examining the value that is specified in the CONNECTREMOTE system option, as follows:

```
%put %sysfunc(getoption(connectremote));
```

or

```
proc options option=connectremote;
run;
```

For example, suppose the output from the %PUT statement shows **`unixhost`**, but you want to define the macro for your Windows machine **`winhost`**:

```
%syslput currentds=ds2003/remote=winhost;
```

As another example, two server sessions are created and the macro variable FLAG must be set in both sessions. The /REMOTE= option is used in the %SYSLPUT statements to direct the correct value to the correct server session.

```
signon task1 sascmd="sas";
signon task2 sascmd="sas";

%syslput flag=1/remote=task1;
/* NOTE: Without the /REMOTE= option in the previous statement,
the FLAG variable would be defined in the TASK2 session,
because it was the session most recently accessed with the
previous SIGNON statement. */
rsubmit task1;
  %put flag on task1 is &flag;
endrsubmit;
%syslput flag=2/remote=task2;
/* NOTE: Without the /REMOTE= option in the previous statement,
the FLAG variable would be defined in the TASK1 session,
because it was the session most recently accessed with
the previous RSUBMIT statement. */
rsubmit task2;
  %put flag on task2 is &flag
endrsubmit;
```

What Session Are Macro Variables Set In When Using the CALL SYMPUT Routine?

Macro variables are set in the server session when you use the CALL SYMPUT routine in a DATA _NULL_ DATA step because the DATA step CALL SYMPUT statements are not macro statements. A sample macro that creates the macro variables in the server session follows:

```
%macro nullds;
rsubmit;
  data _null_;
    call symput('abc','abc');
    call symput('one','1');
    call symput('time',"%sysfunc(putn(%sysfunc(time()),time.))");
  run;
  %nrstr(%%)put abc=&abc one=&one time=&time;
 endrsubmit;
 %mend;
 %nullds;
```

How Do I Know What Session a Macro Is Executed In?

Why does a macro always execute in a client session but sometimes not in a server session?

Even if all the following conditions are met, a macro might *not* execute in the server session, as expected.

- SAS is run in line mode
- the macro is the last line of an RSUBMIT block
- the macro invocation does *not* end with a semicolon (;).

For example, you can invoke the MYDATE macro (without a semicolon) in a client session, as follows:

```
%mydate
```

If you execute SAS in full-screen or DMS mode, invoking MYDATE (with or without the semicolon) in a remote submit will execute correctly.

However, if you execute SAS in line mode, and if MYDATE is defined in the server session and you are remote submitting the invocation of MYDATE as the final line in an RSUBMIT block, you must use the semicolon to delimit the macro invocation, as follows:

```
RSUBMIT;
   %MACRO MYDATE;
      %PUT &SYSDATE;
   %MEND MYDATE;
   %MYDATE;           /* must use semicolon here */
ENDRSUBMIT;
```

When you execute SAS in line mode, the RSUBMIT statement indicates that all subsequent statements are to be processed in the server session. SAS/CONNECT searches the beginning of each statement for the occurrence of the ENDRSUBMIT statement, which indicates that statement processing in the server session should end. The semicolon delimits the end of each statement, except a comment. If the semicolon is omitted, the beginning of the next statement cannot be detected, which causes the ENDRSUBMIT statement to be ignored. The ENDRSUBMIT statement will be sent to the server session along with the macro invocation. The client session will continue to search for an ENDRSUBMIT statement.

In order to execute the remote submit block, including the macro invocation, enter another ENDRSUBMIT statement. Issuing the second ENDRSUBMIT causes the remote submit block to execute. Although the second ENDRSUBMIT is successful, the first ENDRSUBMIT produces the following error message:

```
Statement is not valid or it is used out of proper order.
```

Why Does the Error "Apparent symbolic reference USER1 not resolved" Occur?

This error occurs when a macro variable has not been defined in the SAS session where it is used. This error can occur in a server session when a %LET statement executes in the client session. You can use %NRSTR and %SYSLPUT to ensure that the macro is defined in the server session. You can also put the %LET statement in a macro definition so that the macro variable will be defined in the server session when the macro is invoked.

In the following code example, all the %LET statements are specified in an RSUBMIT block. The &USER1 macro variable is assigned in the client session rather than in the server session, as intended. This problem can be fixed by using the %SYSLPUT or %NRSTR statements. The &USER2 macro variable is assigned in the server session because it is contained in a macro definition in the RSUBMIT block.

```
%macro client;
   RSUBMIT;
      %let user1 = %sysget(LOGNAME);

      %macro remote;
         %global user2;
         %let user2 = %sysget(LOGNAME);
      %mend remote;
      %remote
      data _null_;
        put "user 1 = &user1";
        put "     2 = &user2";
        run;
   ENDRSUBMIT;
%mend client;
%client
```

The %LET statement for USER1 is executed in the client session, but the DATA step is executed in the server session. If the USER1 macro variable has not been previously defined, the following error message will be displayed:

```
Apparent symbolic reference USER1 not resolved.
```

You can set the MLOGIC system option to trace macro processing and to write trace output to the SAS log. Statements that generate a log message are processed in the client session. Statements that do not generate a log message are processed in the server session. For details about MLOGIC, see *SAS Language Reference: Dictionary*.

How Do I Avoid Spacing Problems When Using Semicolons in Macro Values?

My macro-generated RSUBMIT contains several %LET statements whose semicolons are followed with spaces. How can I include semicolons in my macro values and have the value concatenated correctly?

The code follows:

```
%MACRO SETPATH;
    rsubmit;
      %nrstr(%%let PATH1 = c:\winnt\system32%%str(;););)
      %nrstr(%%let PATH2 = c:\winnt%%str(;););)
      %nrstr(%%let PATH3 = c:\bin;)
      %nrstr(%%let PATH  = &PATH1.&PATH2&.&PATH3)
      %nrstr(%%put PATH  = &PATH)
    endrsubmit;
    %MEND;
    %SETPATH
```

The content of the SAS log follows:

```
NOTE: Remote submit to MAINPC commencing.
    1     %let PATH1 = c:\winnt\system32%str(;
    2    );
    3     %let PATH2 = c:\winnt%str(;
    4    );
    5     %let PATH3 = c:\bin;
    6     %let PATH = &PATH1.&PATH2&.&PATH3;
    7     %put PATH = &PATH;
    PATH = c:\winnt\system32; c:\winnt; c:\bin
    NOTE: Remote submit to MAINPC complete.
```

Notice that the semicolons in the PATH macro variables are followed by extraneous spaces.

Because a semicolon is used to terminate a SAS statement, an %STR(;) statement within an %NRSTR statement causes problems when SAS/CONNECT parses the lines and buffers them before sending them to the server session.

To recover from the problem, modify the macro by using %SYSLPUT to submit the SEMICOLON macro variable to the server session. Execution of &SEMICOLON in the server session causes a semicolon to be appended to each %LET statement. The modified code follows:

```
%MACRO SETPATH;
  %syslput semicolon=%nrstr(;);
  rsubmit;
     %nrstr(%%let PATH1 = c:\winnt\system32&SEMICOLON;)
     %nrstr(%%let PATH2 = c:\winnt&SEMICOLON;)
     %nrstr(%%let PATH3 = c:\bin;)
     %nrstr(%%let PATH  = &PATH1.&PATH2&.&&PATH3;)
     %nrstr(%%put PATH  = &PATH;)
  endrsubmit;
  %MEND;
  %SETPATH
```

Using the SEMICOLON macro variable, the %SETPATH macro prints the &PATH macro value without spaces.

The log follows:

```
NOTE: Remote submit to MAINPC commencing.
   8      %let PATH1 = c:\winnt\system32&SEMICOLON
   9      %let PATH2 = c:\winnt&SEMICOLON
   10     %let PATH3 = c:\bin;
   11     %let PATH = &PATH1.&PATH2&.&PATH3;
   12     %put PATH = &PATH;
   PATH = c:\winnt\system32;c:\winnt;c:\bin
   NOTE: Remote submit to MAINPC complete.
```

Compute Services and Break Windows

Overview of Break Windows

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

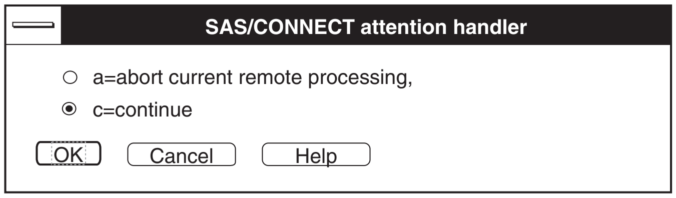
SAS/CONNECT Attention Handler Window

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

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.

Display 11.2 The SAS/CONNECT Attention Handler Window


SAS/CONNECT attention handler

○ a=abort current remote processing,
[✓ c=continue]

The following selections are available in the Attention Handler window:

a
: aborts the statements that are currently being processed on the server 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 local session immediately. △

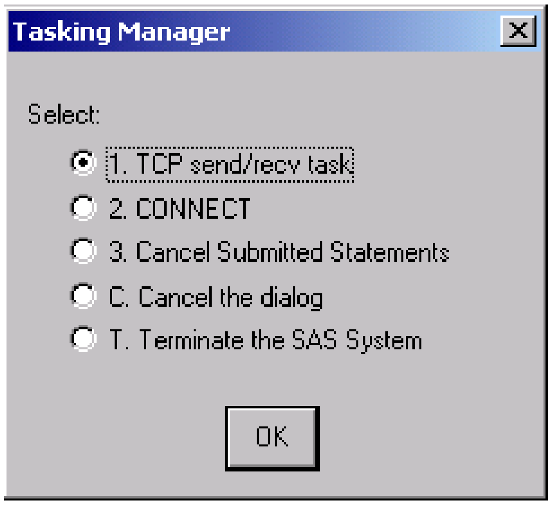
c
: continues the remote job. Select this option if you decide that you do not want to interrupt the remote job.

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:

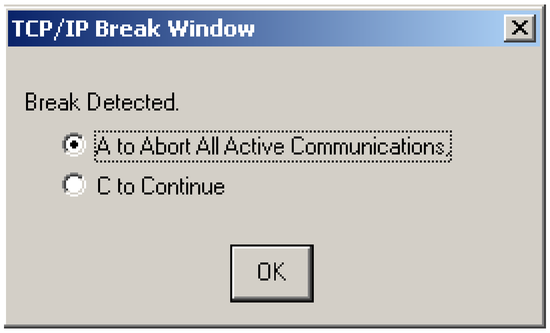
`Ctrl-Break` displays the Tasking Manager window.


Tasking Manager

Select:

[✓ 1. TCP send/recv task]
2. CONNECT
3. Cancel Submitted Statements
C. Cancel the dialog
T. Terminate the SAS System

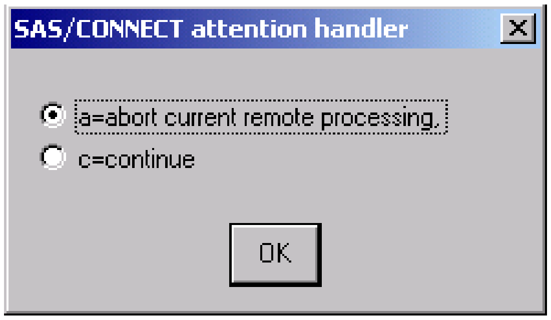
Selecting**`1. TCP send/recv task`** displays the TCP/IP Break window.


TCP/IP Break Window

Break Detected.

[✓ A to Abort All Active Communications]
C to Continue

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


SAS/CONNECT attention handler

[✓ a=abort current remote processing.]
c=continue

CHAPTER 12 Syntax for RSUBMIT Statement and Command

RSUBMIT Statement and Command

Marks the beginning of a block of statements that a client session submits to a server session for processing.

Valid In: Client Session

Syntax

RSUBMIT<*options*>

Options

CONNECTREMOTE=*server-ID*
server-ID

specifies the name of the server session that the statements are executed in. If only one session is active, *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. You can find out which server session is current by examining the value assigned to the CONNECTREMOTE system option.

You can specify *server-ID* using different formats:

❶ *process-name*
process-name is a descriptive name that you assign to the server session on a multi-processor machine when the SASCMD= option is used.

❷ *machine-name*
machine-name is the name of a machine that is running a Telnet daemon or that is running a spawner that is not specified as a service. If the machine name is longer than 8 characters, a SAS macro variable name should be used.

❸ *machine-name.port-name*
machine-name is the name of a server, and *port-name* is the name of the port that the spawner service runs on. If the machine name is longer than 8 characters, a SAS macro variable name should be used.

❹ *machine-name.port-number*
machine-name is the name of a server, and *port-number* is the port that the spawner service runs on.

CAUTION:
Specifying *machine-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 *machine-with-port*.
- □ when used in a LIBNAME statement.

  Instead, use a one-level name or a two-level name, such as *machine-name._ _port-number*.

△

❺ *machine-with-port*
machine-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.

❻ *machine-name._ _port-number*
machine-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.
These examples of specifying *server-ID* correspond to the preceding formats

❶
```
rsubmit emp1 sascmd="!sascmd";
```
❷
```
%let sashost=hrmach1.dorg.com;
rsubmit sashost;
```
❸
```
%let sashost=hrmach1.dorg.com;
rsubmit sashost.sasport;
```
❹
```
rsubmit hrmach1.2267;
```
❺
```
%let sashost=hrmach1.dorg.com 2667;
rsubmit sashost;
```
❻
```
rsubmit hrmach1._ _2267;
```

Alias: CREMOTE=, REMOTE=, PROCESS=

CONNECTWAIT=YES | NO
specifies whether RSUBMIT is to be executed synchronously or asynchronously. Synchronous processing indicates that server processing must be completed before control is returned to the client session.

Asynchronous processing permits multiple server session processes to execute in parallel. Control is returned to the client session immediately after an RSUBMIT begins execution to allow continued processing in the client session and in other server sessions.

If the CONNECTWAIT= option in RSUBMIT is omitted, the value for the CONNECTWAIT= option is resolved as follows:

1 If the CONNECTWAIT= option was specified in the SIGNON statement or for an autosignon (AUTOSIGNON global option is set), that specified value is used.
2 If the CONNECTWAIT global system option is set, the value for the global option is used.

3 If the CONNECTWAIT= option has not been specified in the SIGNON statement or as a system option, the default is to execute synchronously.

Valid values for the CONNECTWAIT= option follow:

YES | Y	specifies that RSUBMITs execute synchronously.
NO | N	specifies that RSUBMITs execute asynchronously.

If CONNECTWAIT=NO is specified, you might also specify the CMACVAR= option. Setting CMACVAR= enables you to programmatically test the status of the current asynchronous RSUBMIT to find out whether it has completed or is still in progress.

When %SYSRPUT executes within a synchronous (CONNECTWAIT=YES) RSUBMIT, the macro variable is defined to the client session as soon as it executes.

When %SYSRPUT is executed within an asynchronous (CONNECTWAIT=NO) RSUBMIT, the macro variable is defined in the client session when a synchronization point is encountered. To override this behavior, use the CSYSRPUTSYNC= option. For more information, see "SYSRPUTSYNC System Option" on page 24. For information about synchronization points, see "Synchronization Points" on page 146.

Note: If CONNECTWAIT=NO is specified and an autosignon is performed, an automatic SIGNOFF will not occur unless PERSIST=NO is also specified. △

Alias: CWAIT=, WAIT=

Default: YES

CMACVAR=*value*

specifies the name of the macro variable to associate with the current RSUBMIT block. Specifying CMACVAR= in an individual RSUBMIT associates the macro variable only with that RSUBMIT. If multiple asynchronous RSUBMITs execute in the same server session and each RSUBMIT contains a CMACVAR= specification, each macro variable will be associated with the respective RSUBMIT block.

Note: If the RSUBMIT command fails because of incorrect syntax, the macro variable is not set. △

Except for the syntax failure condition, the macro variable is set at the completion of the RSUBMIT block.

Valid values for CMACVAR= are:

0	the RSUBMIT is complete.
1	the RSUBMIT failed to execute.
2	the RSUBMIT is still in progress.

Note: If a synchronous RSUBMIT (CONNECTWAIT=YES) is issued while an asynchronous RSUBMIT (CONNECTWAIT=NO) is still in progress, all spooled log and output statements are merged into the client Log and Output windows. Then the asynchronous RSUBMIT resumes processing as if it were synchronous. Control returns to the client session after the synchronous RSUBMIT has completed.

To prevent a conversion from asynchronous to synchronous behavior, use the CMACVAR= option in the RSUBMIT statement (or the SIGNON statement) so that you can check the progress of RSUBMIT without causing it to execute synchronously. △

Alias: MACVAR=

CONNECTSTATUS=YES | NO

specifies the setting for the display of the status window for any file transfers in the current RSUBMIT.

Valid values for this option follow:

YES | Y	The Transfer Status window *is* displayed for file transfers within the current RSUBMIT. For more information, see "Transfer Status Window" on page 219.

NO | N	The Transfer Status window is not displayed for file transfers within the current RSUBMIT.

If the CONNECTSTATUS= option in RSUBMIT is omitted, its value is resolved as follows:

1 If a value for the CONNECTSTATUS= option was specified in the SIGNON statement, the value for the CONNECTSTATUS option is used.

2 If a value for the CONNECTSTATUS global system option was specified, the value for the global 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.

Default: NO for asynchronous RSUBMITs.

SYSRPUTSYNC=YES | NO

enables you to override the default behavior of an asynchronous RSUBMIT so that you can force the %SYSRPUT macro variables to be set in the client session when executed rather than waiting until a synchronization point is encountered. For details about synchronization points, see "Synchronization Points" on page 146.

Note: SYSRPUTSYNC is valid only for an asynchronous (CONNECTWAIT=NO) RSUBMIT. Otherwise, the option is ignored. △

Valid values for this option follow:

YES | Y	the default asynchronous RSUBMIT behavior is overridden. YES forces the macro variables to be defined when %SYSRPUT executes. Because RSUBMIT executes asynchronously, the timing of the definition of the macro variable and its setting is unpredictable. However, the variable will be set before the asynchronous RSUBMIT completes, at the latest.
NO | N	macro variables are set in the client session when a synchronization point is encountered.

If you run multiple RSUBMIT blocks asynchronously, the value that is assigned to the SYSRPUTSYNC= option in the subsequent RSUBMIT block overrides the value that is assigned to the SYSRPUTSYNC= option in the former RSUBMIT block that is still running. For example, if the first RSUBMIT block sets SYSRPUTSYNC=YES and the second RSUBMIT block assumes the default value of NO, the SYSRPUTSYNC=NO overrides SYSRPUTSYNC=YES. SYSRPUTSYNC=NO causes macro variables to be set when a synchronization point is encountered rather than when %SYSRPUT is executed. The problem becomes apparent only if the values of consecutively executed SYSRPUTSYNC= options conflict with the other. For an example of how to prevent SYSRPUTSYNC= option overrides, see "Example 8: Forcing Macro Variables to Be Defined When %SYSRPUT Executes" on page 162.

SYSRPUTSYNC is also a global option. If the SYSRPUTSYNC global option is set, the SYSRPUTSYNC= option that is specified in the RSUBMIT statement or command takes precedence over the global system option. If SYSRPUTSYNC is set as a global option and SYSRPUTSYNC= is not set as an option in RSUBMIT, the global option setting will apply to the RSUBMIT block. For more information, see "SYSRPUTSYNC System Option" on page 24.

Alias: CSYSRPUTSYNC=

USERNAME=*user-ID* | _PROMPT_

For the autosignon feature only, specifies the user ID to be used when connecting to a server session. Valid values that can be assigned to USERNAME= are:

user-ID	For details about a valid user ID, see "User Name and Password Naming Conventions" on page 142.
PROMPT	specifies that SAS prompt the user for a valid username. This value enforces security.

Alias: USERID=, USER=, UID=

PASSWORD=*password* | _PROMPT_ | *"encoded-password"*

For the autosignon feature only, specifies the password to be used when connecting to a server. The operating environment that the server session runs in can affect password naming conventions. For details about password-naming conventions that are imposed by the operating environment, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Valid values for PASSWORD are:

password

must be a valid SAS name that is 1 to 8 characters in length. 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.

"*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 PROC PWENCODE statement. For details, see the *Base SAS Procedures Guide*.

Example code for obtaining an encoded password:

```
 proc PWENCODE in="srvmach";
run;
{sas001}c2VydmlhY2g=
```

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=

CONNECTPERSIST=YES | NO

specifies whether a connection persists or is automatically terminated with an implicit SIGNOFF after the RSUBMIT has completed.

YES | Y	A connection to the server session persists, which means that a signoff is not automatically performed after the RSUBMIT has

completed. CONNECTPERSIST maintains the connection for RSUBMITs.

NO | N	A connection to the server session does not persist. A signoff is automatically performed after the RSUBMIT has completed. A NO value requires that you explicitly sign on before subsequent RSUBMITs.

Alias: CPERSIST=, PERSIST=

Default: YES

CONNECTPERSIST is also a global option. If the CONNECTPERSIST global option is set, the CONNECTPERSIST= option that is specified in the RSUBMIT statement or command takes precedence over the global system option. For more information, see "CONNECTPERSIST System Option" on page 15.

CSCRIPT=*value*
For the autosignon feature only: specifies the script file to use during an autosignon using RSUBMIT. *Value* can be a fileref or a quoted, fully-qualified pathname.

Alias: SCRIPT=

SASCMD="*SAS-command*" | "!sascmd" | "!sascmdv" | "*host-command-file*"
For the autosignon feature only: one of the forms of this command can be used to invoke the server session on the same multi-processor machine (SMP). This option is most useful when using SMP hardware.

"*SAS command*"

- □ For OpenVMS Alpha, UNIX, and Windows: specifies the command that is used to invoke the server session.
- □ For z/OS: specifies a colon that is followed by any SAS invocation options.

  Example:

```
sascmd=":ls=256"
```

!sascmd
For OpenVMS Alpha, UNIX, and Windows: starts a server session by using the same command that was used to invoke the client session.

!sascmdv
For OpenVMS Alpha, UNIX, and Windows: starts a server session by using the same command that was used to invoke the client session and writes the SAS invocation to the log.

SASCMD= is also a global option. If the SASCMD= global option is already set, the SASCMD= option that is specified in the RSUBMIT statement or command takes precedence over the global system option. For more information, see "SASCMD= System Option" on page 20.

An example of a typical setting for the SASCMD= option follows:

```
SASCMD="sas"
```

As another example, commands that contain spaces must be enclosed in double quotation marks.

```
rsubmit rmthost sascmd='"c:\Program Files\SAS\SAS System\9.1\sas.exe"';
```

In order to execute additional commands prior to SAS invocation, you might write a command file that is specific to your operating environment. File-naming conventions conform to the requirements of the operating environment, as follows.

Operating Environments	**Filename Extensions**
OpenVMS Alpha	.com
UNIX	.ksh*
Windows	.bat, .cmd

* The UNIX file name extension depends on the specific shell that is being used; for example, Korn shell, C shell, or Bourne shell.

Note: The SASCMD= option does not support z/OS command files. △

The TCP/IP access method adds options, such as -DMR, to the server's SAS command. For Windows, the TCP/IP access method also appends the following options:

- -COMAMID TCP
- -ICON
- -NOLOGO
- -NOTERMINAL

An example of creating a command file named *mysas.bat* for Windows follows:

```
cd "C:\Program Files\alpair\SAS\V9.1"
mkdir mywork
sas %* -nosyntaxcheck -work "mywork"
```

*%** adds the appended TCP/IP access method options to the SAS invocation in *mysas.bat*.

To execute the command file, specify its name as the value for SASCMD=.

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

CAUTION:

OpenVMS Alpha operating environment only: If the NODETACH system option is set, and if multiple server sessions are running on OpenVMS Alpha and you observe degraded performance, you will be alerted to the condition by an error message. For example:

```
ERROR: Process quota exceeded.
ERROR: Cannot spawn subprocess. Check process limit quotas and privileges.
```

NODETACH causes a signon to occur in a subprocess of the parent's process, which can drain resources. If NODETACH is set, try setting the DETACH system option, which causes signons to occur as detached processes rather than as subprocesses. For more information, see the NODETACH system option in the *SAS Companion for OpenVMS Alpha*.

To improve performance with NODETACH set, ask your system administrator to set the following resources to the specified values for each signon to a server:

Table 12.1 OpenVMS Operating Environment Resource Values

User Account Resource	**Minimum Value**
Paging file quota	40000
Buffered I/O byte count quota	13000
Open file quota	65

User Account Resource	Minimum Value
Subprocess limit	1
Timer queue entry limit	1 to 8

Signon via the SASCMD= option is not supported when running SAS from a captive OpenVMS account. SASCMD signons require the process in which SAS is running to start a remote SAS session, either in a subprocess or in a detached process. Starting subprocesses is not allowed under a captive account. A detached process that runs under a captive account will not be able to invoke SAS because a captive OpenVMS account is under the control of the login command procedure. The command language interpreter will execute only the commands in your login command procedure and then the process will exit.

The **!sascmdv** value in the SASCMD= option causes the display of a symbol that specifies how the server session was started. You can print the symbol's value by using the **getsym** DATA step function.

```
rsubmit;
   %put %bquote(
   %sysfunc (getsym(SASCMD_2042CF6B)));
endrsubmit;
```

△

LOG=KEEP | PURGE |*filename* | *fileref*
OUTPUT=KEEP | PURGE | *filename* | *fileref*

use only with an asynchronous RSUBMIT (CONNECTWAIT=NO). 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 specified file or fileref. A *backing store* is a SAS utility file that is written to disk in the client SASWORK directory. If used outside an asynchronous RSUBMIT, these options are ignored and the following message is displayed:

```
WARNING: LOG=/OUTPUT= options invalid with synchronous rsubmit.
Options will be ignored.
```

Valid values for both the LOG= and OUTPUT= options follow:

KEEP
: spools log or output lines, as applicable, to the backing store or the client machine for later retrieval with the RGET, RDISPLAY, synchronous RSUBMIT, or SIGNOFF statements. KEEP is the default.

PURGE
: deletes all the log or output lines that are generated by the current server session. PURGE is used for saving disk resources. If you can anticipate a large volume of log data or output data to the backing store that you do not want to receive, and you want to preserve disk space, setting PURGE can be useful.

filename | *fileref*
: specifies a file that is 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. A subsequent RSUBMIT to the same file that you specify by using the LOG= or the OUTPUT= option appends the contents of the current RSUBMIT to the same file.

In order to append log or output lines from multiple RSUBMITs into the same file, use the FILENAME statement to specify a fileref and use the MOD= option to open the referenced file for an append.

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, you will receive a message such as the following:

```
WARNING: The LOG option was used to file log lines for the current RSUBMIT.
There are no log lines for RGET to process.
```

CAUTION:

Do not simultaneously use the asynchronous RSUBMIT and the PROC PRINTTO statement and re-direct output. Re-directing output by using a LOG= or an OUTPUT= option in the asynchronous RSUBMIT statement and by using a locally submitted PROC PRINTTO statement can cause unpredictable results. △

If you use both the asynchronous RSUBMIT and the PROC PRINTTO statements, you might expect that the PROC PRINTTO statement forces data from the server session to be written to the file that is specified in the PROC PRINTTO statement. If that happens, the LOG= or the OUTPUT= option in the RSUBMIT statement is ignored, and no data is written to the backing store or 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 RSUBMIT log or to the PROC PRINTTO log.

NOTIFY=YES|NO

requests the display of a notification message window to report the completion of an asynchronous RSUBMIT. The message includes the server session ID, which is TASK1, in the following example.

```
Asynchronous task TASK1 has completed
```

The display window does not interfere with any other SAS session executions in progress. To acknowledge the message and to close the window, click OK.

Valid values follow:

YES|Y	enables notification.
NO|N	disables notification. This is the default.

The NOTIFY= option is applied only during signon and autosignon. After being set, notification remains in effect for the duration of the server session.

A notification message window is displayed upon completion of an asynchronous RSUBMIT, and for all SAS session execution modes (DMS, line, and batch) that a terminal is attached to. You associate a terminal with a SAS session by setting the TERMINAL option.

If you try to set the NOTIFY= option in a SAS session that has been invoked by using the NOTERMINAL option, the following message is displayed:

```
WARNING: The NOTIFY option is valid only if a TERMINAL is attached to this
SAS session. Option will be ignored.
```

To disable notification during a session, you must sign off the server session and then sign on to the server session again, and either omit the NOTIFY= option or specify NOTIFY=NO in the SIGNON statement.

If you specify the NOTIFY= option in an RSUBMIT that is not executing an autosignon, the following message is displayed:

```
WARNING: The NOTIFY option is applied only during SIGNON,
 but remains in effect for the entire connection until SIGNOFF.
```

If notification is set and you issue statements or commands (such as RGET or SIGNOFF) during the asynchronous RSUBMIT that change processing from asynchronous to synchronous mode, the notification window is *not* displayed.

INHERITLIB=(*client-libref1 <=server-libref1> ... client-librefn <=server-librefn>*)
allows 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 SASWORK library cannot be reassigned in any SAS session, you cannot reassign it in the server session either. △

Example:

```
libname scorcard '.';
rsubmit inheritlib=(scorcard work=cwork);
  libname scorcard list;
  libname cwork list;
  data scorcard.a; x=1; run;
endrsubmit;
proc datasets lib=scorcard; run;
```

SERVER="*Connect-server-definition*"
is used to specify a CONNECT server definition that has been defined in a SAS Metadata Repository. SAS Management Console (SMC) can be used to create, update, and delete server definitions.

Note: Because the CONNECT server definition configures all server options, do not specify any other server options by using the SERVER= option. △

If you have not set global options to specify how to connect to the SAS Metadata Server, a requestor window is displayed in which you can configure the IP address for the server, the port, the protocol, and the user name and password.

Note: The CONNECT Server definition must be enclosed in quotation marks. △

For complete details about creating and populating a SAS Metadata Repository, see the documentation at **support.sas.com/rnd/eai/openmeta**.

SIGNONWAIT=YES|NO
SIGNONWAIT specifies whether implicit signons within an RSUBMIT are executed synchronously or asynchronously during a SAS session. Synchronous processing means that an RSUBMIT to a server session must complete before control is returned to the client session.

YES|Y	specifies that a SAS/CONNECT implicit signon in an RSUBMIT will execute synchronously. This means that a SAS/CONNECT RSUBMIT must be complete before subsequent processing can occur. This is the default.
NO|N	specifies that a SAS/CONNECT implicit signon in an RSUBMIT will execute asynchronously. This means that subsequent processing does not have to wait for a SAS/CONNECT RSUBMIT to complete.

Asynchronous processing permits implicit signons in an RSUBMIT to multiple server sessions to execute in parallel. Control is returned to the client session immediately after an implicit signon in an RSUBMIT when SIGNONWAIT=NO has been specified.

You can use SIGNONWAIT=NO 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 remote submitting units of work to a server session as soon signon is complete.

SIGNONWAIT is also a global option. If the SIGNONWAIT global option is set, the SIGNONWAIT= option that is specified in the RSUBMIT statement or command takes precedence over the global system option. If SIGNONWAIT is set as a global option and SIGNONWAIT= is not set as an option in RSUBMIT, the global option setting will apply to the RSUBMIT block. For more information, see "SIGNONWAIT System Option" on page 23.

Details

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 causes a remote execution, 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 can be used in any type of client session.

You can issue the RSUBMIT command from any of the following:

- the command line of the client session Program Editor window
- in a DM statement, which treats commands as if they were issued from a windowing environment command line.
- 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 processed either synchronously or asynchronously.

synchronous
: Client session control is not returned until the RSUBMIT has completed. Synchronous processing is the default processing 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.

Synchronous RSUBMITs display 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.

Processing Statements in the RSUBMIT/ENDRSUBMIT 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 a DMS session, an interactive line-mode session, or as a non-interactive job. The RSUBMIT and the ENDRSUBMIT statements enable you to include the statements that should be processed in the server session in the same file as the statements that will be processed in the client session. The statements for the server session are enclosed between the RSUBMIT and the ENDRSUBMIT statements. All the other statements in the program are processed in the client session when you execute 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;
```

Note: The DOWNLOAD and the UPLOAD procedures must be executed by using the RSUBMIT command or the RSUBMIT statement. You cannot execute them by using the SUBMIT command. △

Autosignon Feature of RSUBMIT The autosignon feature of RSUBMIT includes an implicit SIGNON (commonly called autosignon) in the absence of a current connection. By default, AUTOSIGNON is not in effect. To activate autosignon, specify AUTOSIGNON. For more information, see "AUTOSIGNON System Option" on page 13. An example of setting the option follows:

```
option autosignon;
```

The RSUBMIT command or statement automatically executes a SIGNON, and uses any SAS/CONNECT global system options along with any connection options that are specified with RSUBMIT. The autosignon signs on to the server session and executes the RSUBMIT. All SIGNON options are also valid for RSUBMIT.

If autosignon is specified for a synchronous or an asynchronous RSUBMIT, the default behavior is to maintain the connection to the server (not to sign off) after the RSUBMIT has completed. However, the CONNECTPERSIST= option in the RSUBMIT statement can be used to override this default behavior.

Any connection information that is specified in RSUBMIT for autosignon is in effect for the entire connection. For example, if you specify CONNECTWAIT=NO in an RSUBMIT that automatically signs on, asynchronous RSUBMITs will be the default for the entire connection. However, the CONNECTWAIT= value can be overridden in individual RSUBMITs.

User Name and Password Naming Conventions Each user name and password is limited to 256 characters that follow these conventions:

- Mixed case is allowed.
- A null value, which is no value, that is delimited with quotation marks is allowed.
- Quotation marks must enclose values that contain one or more spaces.
- Quotation marks must enclose values that contain one or more special characters.
- Quotation marks must enclose values that contain one or more quotation marks.
- Quotation marks must enclose values that begin with a numeric value.
- Quotation marks must enclose values that do not conform to rules for user-supplied SAS names. For details about rules for SAS names, see *SAS Language Reference: Dictionary*.

Examples:

```
user=joe password='Born2run';
user=joe password='' /* null space specified by contiguous quotation marks */;
user='joe black' password='Born 2 run';
user='joe?black' password='Born 2 run';
user='apexdomain\joe' pass='2bornot2b' /* Windows user name */;
user='"happy joe"' pw=_prompt_;
```

```
user=_prompt_;
userid="myuserid" password="{sas001}MVNoYXJl";
```

ENDRSUBMIT Statement

Marks the end of a block of statements that a client session submits to a server session for processing.

Valid In: Client Session

Syntax

ENDRSUBMIT <CANCEL>;

Syntax Description

CANCEL
terminates the block of statements without executing the statements. This option is useful in a 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:

```
dm 'rsubmit';
```

or

```
rsubmit;
```

The server session processes the statements between either of these statements and the ENDRSUBMIT statement.

You 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: from a DMS session, an interactive line-mode session, or a non-interactive job. The RSUBMIT and ENDRSUBMIT statements enable you to include in the same file, statements that are processed in the client session and statements that are processed 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 processed in the client session when you execute the program. The following template is used to build a file that includes statements for both the server and client sessions in the same program:

```
statements for client session
rsubmit;
   statements for server session
endrsubmit;
more statements for client session
```

RDISPLAY Command and 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-ID*
server-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=, REMOTE=, PROCESS=

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 the DM statement. The RDISPLAY statement can be used in any type of client session.

RGET Command and 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-ID*
server-ID
specifies the name of the server session that generated the spooled log and output that you want to retrieve. 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 current by examining the value assigned to the CONNECTREMOTE system option.

Alias: CREMOTE=, REMOTE=, PROCESS=

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 statements are *not* merged into the client Log and Output windows, but instead, by default, they are spooled until retrieved at a later time.

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 processing as if it were submitted synchronously. Control is not returned to the client session until the RSUBMIT has completed.

If you do not want the RSUBMIT to become synchronous, but you want to check its progress, use the CMACVAR= option in the SIGNON or the RSUBMIT 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 "Monitoring MP CONNECT Tasks" on page 113. △

Note: For asynchronous RSUBMITs, the SAS system option _LAST_, which is used to determine the name of the most recently created data set, does not get updated. Also, if RGET is used to change processing modes from asynchronous to synchronous, the system option _LAST_ does not get updated. For more information about _LAST_, see *SAS Language Reference: Dictionary*. △

%SYSRPUT Statement

Assigns a value from the server session to a macro variable in the client session.

Valid In: Server Session

Syntax

%SYSRPUT *macro-variable=value*;

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 will be assigned to the *macro-variable* in the client session.

Details

The %SYSRPUT statement is a macro statement that must be remote submitted to the server session to assign a value that is available in the server session to a macro variable that can be accessed from the client session. The %SYSRPUT statement is similar to the %LET statement because it is used to assign a value to a macro variable. However, the %SYSRPUT statement assigns a value to a variable in the client session, not in the server session where the statement is processed. The %SYSRPUT statement puts the macro variable into the Global Symbol Table in the client session.

A synchronization point identifies the point during an asynchronous RSUBMIT at which the macro variable that is specified in the %SYSRPUT statement is defined to the client session so that users can use it in their client processing.

Synchronization Points

The three possible synchronization points are:

1. when the RGET command is executed.

   At this point, all macro variables that were specified by using %SYSRPUT are defined in the client session and are available for processing.

2. when a synchronous RSUBMIT is started in the same server session that an asynchronous RSUBMIT is already running in.

3. when the SIGNOFF command or the SIGNOFF statement is executed.

   All macro variables that were specified using %SYSRPUT are defined in the client session and are available for processing.

All currently spooled log and output statements are retrieved and merged into the client Log and Output windows. RSUBMIT continues from that point 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 processing up to that point are defined in the client session. However, from that point on, 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 CSYSRPUTSYNC= option in the asynchronous RSUBMIT statement. Macro variables are set at the time of execution rather than at a synchronization point in the client session.

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 non-interactive job.

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 processing in the server session has completed, the job checks the value of the return code that is stored in RETCODE. If server processing is successful, processing continues in the client session.

```
rsubmit;
      proc download data=remote.mydata
         out=local.mydata;
      run;
      %sysrput retcode=&sysinfo;
endrsubmit;

%macro checkit;
   %if &retcode=0 %then %do;
      further processing in client session
   %end;
%mend checkit;
%checkit;
```

A SAS/CONNECT batch (non-interactive) job returns a non-zero system condition code if the SIGNON, RSUBMIT, or SIGNOFF fails. To determine the success or failure of the job steps within an RSUBMIT block in a non-interactive job, use the %SYSRPUT macro statement to check 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 CSYSRPUTSYNC= option is specified in order to set the client 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.

```
rsubmit wait=no csysrputsync=yes;
   %sysrput status=start;
   proc download inlib=sales outlib=tmp
      status=n;
   run;
   %sysrput status=salescomplete;

   proc download inlib=inventory outlib=tmp
      status=n;
   run;
   %sysrput status=inventorycomplete;

   proc upload data=sales.store10 status=n;
   run;
   %sysrput status=storecomplete;
endrsubmit;
```

Example 3: %SYSRPUT

This example shows how to identify the server session that the client session is connected to:

```
rsubmit;
%sysrput rhost=&sysscp;
endrsubmit;
```

%SYSLPUT Statement

Creates a macro variable in the server session.

Valid In: Client Session

Syntax

%**SYSLPUT** *macro-variable=value* </REMOTE=*server-ID*>;

Syntax Description

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 an alphanumeric string that will be assigned to the server *macro-variable*. The string should not contain nested quotation marks.

/REMOTE=*server-ID*
specifies the name of the session that the macro variable will be created in. If only one session is active, the *server-ID* can be omitted. For multiple server sessions that 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 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.

Details

Note: %SYSLPUT requires that the client and server sessions run SAS Version 8 or later. △

The %SYSLPUT statement is a macro statement that is submitted in the client session to assign a value that is available in the client session to a macro variable that can be accessed from 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.

The %SYSLPUT statement is similar to the %LET statement in its assignment of a value to a macro variable. However, the %SYSLPUT statement makes the assignment to a macro variable in the server session (not in the client session). The %SYSLPUT statement puts the macro variable into the current referencing environment of the server session.

Example 1: %SYSLPUT

The following example sets the macro variable FLAG to 1 in the current server session.

```
%syslput flag=1;
```

Example 2: %SYSLPUT

%SYSLPUT enables you to dynamically assign values to variables that are used by remotely executed macros. The macro statement %SYSLPUT is used to create the macro variable REMID in the server session with 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 determine which operating system-specific library assignment should be used by 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: %SYSLPUT

The optional /REMOTE option in the %SYSLPUT statement requires that any value that contains forward slashes should be quoted with the %BQUOTE macro function. The %BQUOTE function masks a character string or a resolved value of a text expression during execution of a macro or macro language statement.

The following example uses the %BQUOTE function to mask forward slashes that are used in a UNIX pathname that is assigned in the %SYSLPUT statement.

```
  2? %let pathineed=/abc/xyz;
  3? %syslput pathineed=%bquote(&pathineed);
  4? rsubmit;

NOTE: Remote submit to apex commencing.

  5? %put &pathineed
  5? endrsubmit;

1    %put &pathineed
/abc/xyz
NOTE: Remote submit to apex complete.
```

WAITFOR Statement

Causes the client SAS session to wait for the completion of one or more asynchronously executing tasks that are already in progress.

Valid In: Client Session

Syntax

WAITFOR <_ANY_|_ALL_> *task1* ... *taskn* <TIMEOUT=*seconds*>;

Syntax Description

ANY
makes the client session wait for the completion of ANY of the specified tasks (a logical OR of the completion task states).

ALL
makes the client session wait for the completion of ALL of the specified tasks (a logical AND of the completion task states).

task1...taskn
identifies one or more asynchronous tasks to be completed. The task name is the server ID that is associated with the CONNECTREMOTE= option when the RSUBMIT was issued.

TIMEOUT=*seconds*
allots the interval, in seconds, to wait for one or more asynchronous tasks to complete processing. If the specified tasks have not finished processing by time-out, the WAITFOR statement is terminated, control is returned to the client session, and the asynchronous processes continue to execute until they are completed. The SYSRC system macro variable will have a non-zero status.

If the specified tasks finish processing before time-out, the WAITFOR statement returns control to the client session as soon as the specified tasks finish.

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 returning control to the client session. △

Default: 0

Details

The WAITFOR statement is used to make the current SAS session wait for the completion of one or more tasks that are already in progress as specified by the options _ANY_ or _ALL_. WAITFOR synchronizes dependent tasks. You can use WAITFOR only for asynchronously executing tasks (for example, an RSUBMIT that is executed with the CONNECTWAIT= option set to NO). 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 current 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 session will wait for any of the listed tasks to complete before resuming. This is *not* an error condition.

If more than one task is specified, and the _ANY_ option is specified, 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.

Example 1: WAITFOR

The following example shows the suspension of the current session until both tasks have completed or 300 seconds (5 minutes) pass, whichever occurs first.

```
waitfor _all_  remhost printjb  timeout=300;
```

This statement causes the client session to wait for the REMHOST and the PRINTJB tasks to finish. Both tasks must complete within the allotted time or the time must expire before the WAITFOR statement returns control to the client session. If time expires before the completion of both tasks, control is returned to the current session and the asynchronous tasks continue to execute. The SYSRC global macro variable can be queried to detect this condition. For details, see *SAS Macro Language: Reference*. Alternatively, if you specified macro variables for the REMHOST and PRINTJB tasks using the CMACVAR option in the RSUBMIT statement, you could query those macro variables for status information.

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;
```

Because the processing of these tasks is not restricted to a time limit, the client session will be suspended until one of the specified tasks completes. Upon completion of either task, the WAITFOR statement returns control to the current session.

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
gives status information about all current tasks.

task
gives status information about 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.

Details

The LISTTASK statement lists information about a single active task by name or about all tasks in the current session. If neither a task name nor _ALL_ is specified, information about all current tasks is listed.

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: 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.

Details

The KILLTASK statement enables users to terminate one or multiple tasks or server sessions that are executing asynchronously. The KILLTASK statement is useful only for an asynchronous RSUBMIT, which is designated by the CONNECTWAIT=NO option.

Note: KILLTASK should be used for asynchronous tasks that seem to be hung or 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 flushed to the parent Log and Output windows. Prior to flushing the data, KILLTASK issues this message:

```
NOTE: Process TASK1 was terminated by KILLTASK statement.
```

KILLTASK removes the specified task from the list of active tasks, thereby removing it from LISTTASK output. If KILLTASK is issued for a completed task, a 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 cleans up the WORK library of the server session.

CHAPTER 13 Examples Using Compute Services

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

Purpose

Suppose you want to asynchronously submit a lengthy SAS program to a server session for execution. The program, which is expected to run for a long time, calculates summary statistics from the variables in a large SAS data set and downloads the summary statistics to your client session. In addition, you want to define the macro variable REMSTATUS to provide the status of the server task and file all of the log lines to the REMLOG fileref.

Type the following program in the Program Editor window of your client session.

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;
```

To execute the program in the server session, type **SUBMIT** on the command line in the Program Editor window or press the SUBMIT function key.

Example 2: Administering Server Data Sets from a Client

Purpose

While working on a client machine, you can use Compute Services to perform administration tasks on data sets on a server machine.

The following example administers password protection to the TASKLIST data set and backs up a data set that is named CURRENT.

Program

```
rsubmit;
   proc datasets lib=tsolib;
      /*************************************/
      /* Add password SESAME to server      */
      /* data set TASKLIST.                 */
      /*************************************/
   modify tasklist (alter=sesame);
   run;

      /*************************************/
      /* Maintain a week's worth of backup  */
      /* copies of data set CURRENT.        */
      /*************************************/
   age current backup1 - backup7;
   run;
   quit;
endrsubmit;
```

Example 3: Using the MACVAR= Option with MP CONNECT

Purpose

The following example enables you to remote 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 not complete */

%get_results_when_complete;
```

Example 4: Using 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 many different ways.

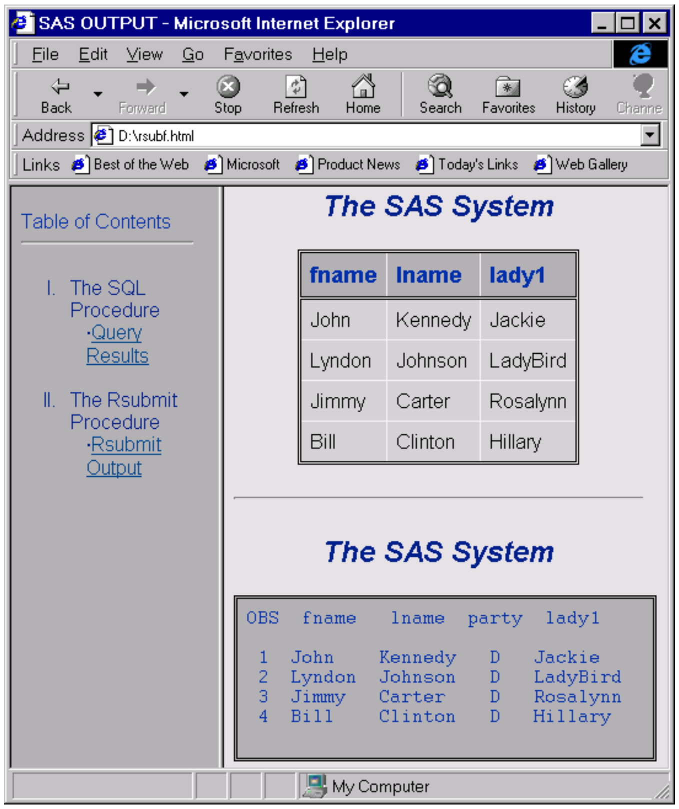
The following program creates a SAS data set and a SAS data view in a server session, which contains information about U.S. Presidents and generates ODS output. The first half of this example creates HTML from the SAS data set and view. The second half uses ODS to create a SAS data set from the SAS data view.

Program

```
rsubmit;

   data presidnt;
      length fname lname $8 party $1 lady1 $10;
      input fname lname party year_in lady1;
   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 W Bush R 2002 Laura
   ;
   run;

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

endrsubmit;

   /* Use ODS to create HTML from the output */

filename rsub "rsub.html" mod;
filename rsubc "rsubc.html" mod;
filename rsubf "rsubf.html" mod;
ods html
   file=rsub;
   contents=rsubc;
   frame=rsubf;

   /* Remote SQL PassThru to SQL view */
```

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

   /* mix remote-submitted SQL with client SQL */
title 'RSPT: Republicans';
rsubmit;
   proc sql nocheck;
     select fname,lname,lady1
        from presidnt
        where party='R';
quit;
endrsubmit;

ods html close;

   /* Use ODS to create a SAS data set */
ods output output="rdata";

rsubmit;
   proc print data=democrat;
   run;
endrsubmit;
```

Display 13.1 SAS Output Window


SAS OUTPUT - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channe
Address D:\rsubf.html
Links Best of the Web Microsoft Product News Today's Links Web Gallery
Table of Contents
I. The SQL Procedure ·Query Results
II. The Rsubmit Procedure ·Rsubmit Output
My Computer

The SAS System

fname	lname	lady1
John	Kennedy	Jackie
Lyndon	Johnson	LadyBird
Jimmy	Carter	Rosalynn
Bill	Clinton	Hillary

The SAS System

```
OBS  fname    lname   party  lady1

 1  John    Kennedy    D    Jackie
 2  Lyndon  Johnson    D    LadyBird
 3  Jimmy   Carter     D    Rosalynn
 4  Bill    Clinton    D    Hillary
```

Example 5: Using 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

```
/* Global SAS system option SASCMD starts an MP CONNECT server session. */
option autosignon=yes;
option sascmd="!sascmd";

/* Remote submit first task. */
/* Sort the first data set as one task. */
/* SIGNON performed automatically by RSUBMIT. */
rsubmit process=task1 wait=no;
libname mydata '/project/test1';

      proc sort data=mydata.part1;
     by x;
run;
endrsubmit;

/* Remote submit second task. */
/* SIGNON performed automatically by RSUBMIT. */
rsubmit process=task2 wait=no;
libname mydata '/project/test2';

   /* Sort the second data set as one task. */
   proc sort data=mydata.part2;
     by x;
run;
endrsubmit;

/* Wait for both tasks to complete. */
waitfor _all_ task1 task2;

/* Merge the results and continue processing. */
libname mydata ('/project/test1' '/project/test2');
data work.sorted;
  merge mydata.part1 mydata.part2;
run;
```

Example 6: Using MP CONNECT with Piping

Purpose

In the following 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

```
/* -----------  DATA Step - Process P1  ----- */
signon p1 sascmd='!sascmd';
rsubmit p1  wait=no;

libname outLib sasesock ":pipe1";

/* create data set - and write to pipe */
data outLib.Intermediate;
   do i=1 to 5;
       put 'Writing row ' i;
       output;
   end;
run;
endrsubmit;
rdisplay p1;

/* -----------  DATA Step - Process P2  ----- */

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

libname inLib sasesock ":pipe1";
libname outLib "d:\temp";

data outLib.Final;
set inLib.Intermediate;
   do j=1 to 5;
        put 'Adding data ' j;
        n2 = j*2;
        output;
   end;
run;
endrsubmit;
rdisplay p2;
/* -------------------------------------------- */
```

Example 7: Preventing Pipes from Closing Prematurely

Purpose

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

In the following 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 prior to 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: Forcing 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 global option is specified, the locally set value that is specified in the CSYSRPUTSYNC= option takes precedence. If CSYSRPUTSYNC= in an RSUBMIT block is omitted, the global value 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 global 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 -unbuflog -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: Graphics Processing on the Server

Purpose

If you have SAS/GRAPH software installed on both your client and server machines, you can submit graphics programs from your client session to a server session, execute the procedure in the server session, and display the graphics output in the client session (or on a device that is attached to your client machine). This link is especially useful when you want to generate graphics in your client session by using a large database that is accessible from the server session.

The GRLINK driver is a special driver that is available with SAS/CONNECT. You must always use the GRLINK driver in the server session when using the link to display server machine graphics in your client session.

If you frequently use the link for server graphics processing, consider specifying the GRLINK device driver in a script file (if you use a script file with the SIGNON command). To do this, include the driver specification for the server machine in the TYPE statement that invokes the server session. In the following program, if you are using TSO by means of a TCP/IP connection, change the TYPE statement in the script file to the following:

```
type
"sas options('comamid=tcp device=grlink dmr')";
```

By changing the TYPE statement in the script file each time that you use the SIGNON command, you automatically specify the GRLINK driver in the server session. The GRLINK driver is specified in the sample scripts that are provided with your SAS software.

The following program uses the RSUBMIT command to submit SAS statements, which include any LIBNAME statements that are needed in the server session. When the SAS/GRAPH procedure runs in the server session, the output is displayed at the client session or on an attached device (based on the driver that you specified in your client session). Make sure to specify the GOPTIONS DEVICE=GRLINK driver as shown in step 3.

Program

```
❶
goptions device='';

❷
rsubmit;

   proc sort data=master.bg_reserve out=tmp;
      by origin rental_type;
   run;
   proc summary data=tmp vardef=n noprint;
      by origin rental_type;
      output out=temp_rental;
   run;

❸
   goptions device=grlink ftitle=centx
   ftext=simplex htitle=2;

   title 'Rental Types by Franchise';
   pattern value=solid color=blue;

   axis1 label=('Franchise')
   order=
   ('ATLANTA' 'CHICAGO' 'LOS ANGELES'
      'NEW YORK' 'TORONTO')
   width=3;
   axis2 label=none width=3;
   axis3 label=none
   order=0 to 1000 by 100 width=3;
   proc gchart data=temp_rental;
      label rental_type='00'x;
      label origin='00'x;
      hbar rental_type /  frame
      sumvar= _freq_
      maxis=axis2
      raxis=axis3
      minor=0
      nostats
      group=origin
      gaxis=axis1
      discrete;
   run;
   quit;

endrsubmit;
```

❶ Specify an EGA graphics adapter to display the server session graph in your client session. You can specify the name of the graphics driver for your client machine display or its attached hard-copy device. To get a complete list of values for the DEVICE= option, run the GDEVICE procedure in your client session. This example sets the device option to a null value so that the default device will be used.

Note: A null value is specified by using two single quotation marks. △

❷ Remote submit procedures to preprocess data and graphics procedures to the server session.

❸ Specify the GRLINK device driver so that commands to draw the graph will be sent over the network to the client session.

When using the link to display server session graphs, you can use any graphics procedure on the server machine (including the GREPLAY procedure) and any graphics device driver on the client machine.

The GRLINK server machine driver uses the attributes of the driver that is specified in the client session when selecting default colors, character sizes, and other attributes. For example, if you specify DEVICE=PSCOLOR in your client session, the GRLINK driver uses the default colors of the PSCOLOR driver, but if you specify the printer driver DEVICE=PCL5 in your client session, the GRLINK driver uses only black as a foreground color.

Note the following reminders when using the link for graphics:

- Do not specify GOPTIONS NODISPLAY in the program that you submit to the server session. If the GRLINK driver is on the server machine, the option is not supported.
- Do not specify DEVICE=GRLINK in your client session. The GRLINK driver can only be specified on the server machine. In your client session, you can specify only a device driver that is available with SAS/GRAPH on that machine.
- You can use hardware options, such as NOCHARACTERS, only on the client machine. You cannot use hardware options that are not available with your client machine hardware configuration even though the options are supported on the server machine.
- To use the CBACK= or the ROTATE= option, you must specify it in your client session program, not in the program that you are submitting to the server session. If you use the CBACK= or the ROTATE= option in the program that is submitted to the server session, the option is accepted but has no effect.
- To use the GREPLAY procedure through the link, you must use the NOFS option in the PROC GREPLAY statement.
- Every time you generate graphics output in the client session, it is stored temporarily, while running the same SAS session, in a catalog called GSEG in the WORK library of the client session. Later, displays of the same graphics output can be generated from this catalog. Copy this catalog to a permanent location if you want to retain a copy after your current session ends.

You can also transfer catalog entries that contain graphics output by using the UPLOAD and DOWNLOAD procedures, as described in “Example 3.4: Using the ENTRYTYPE= Option in Two SELECT Statements in PROC DOWNLOAD” on page 258.

Example 10: Using Server Software from a Client Session

Purpose

Some software might not be available on every machine at your computing site. In addition, the software that is available on a server machine might perform some tasks better than the software that is available on your client machine. From a client session, you can use compute services to use software that is available on a server machine.

The following program assumes that SAS/STAT is licensed only on the server machine. 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;
```

CHAPTER 14 Syntax for Remote SQL Pass-Through (RSPT)

RSPT Statements

Statements used for Remote SQL Pass-Through.

Valid In: Client Session

Syntax

❶

CONNECT TO *dbms-name* <AS *alias*> <(*dbms-argument-1=value* ... <*dbms-argument-n=value*>)>;

SELECT . . . FROM CONNECTION TO *dbms-name* | *alias* (*dbms-query*);

EXECUTE (*SQL-statement*) **BY** *dbms-name* | *alias*;

DISCONNECT FROM *dbms-name* | *alias*;

❷

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-n=value*>)>);

SELECT . . . FROM CONNECTION TO REMOTE | *alias* (*dbms-query*);

EXECUTE (*SQL-statement*) **BY** REMOTE | *alias*;

DISCONNECT FROM REMOTE | *alias*;

Syntax Description

The REMOTE engine supports the SQL procedure's Pass-Through Facility. *Remote SQL Pass-Through* (RSPT) enables you to pass SQL statements to a remote SAS SQL processor or to a DBMS through a SAS/SHARE server or to a SAS/CONNECT single-user server.

❶ The SQL syntax for the SQL procedure Pass-Through (SPT) facility consists of three statements and a FROM-clause component.

❷ The SQL syntax for the Remote SQL Pass-Through (RSPT) facility is similar to the SPT but must also include the server ID.

CONNECT TO REMOTE <AS *alias*>
connects to a remote DBMS or to remote SAS data through a SAS server. This statement is required (RSPT does not support implicit connection). You can establish multiple connections to the same server by specifying different DBMS= values. You can also connect to more than one server at a time.

Note: The term server refers to the SAS/CONNECT single-user server and the SAS/SHARE multi-user server. △

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 may 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

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 you can also RSUBMIT queries to achieve the same goals.) For example, if you specify

```
select employee_title as title, avg(employee_years),
   freq(employee_id)
      from sql.employee
      group by title
      order by title;
```

where SQL is the libref for a remote SAS library that is accessed through a SAS/CONNECT or a SAS/SHARE server, each row of the table EMPLOYEE must be returned to your client session in order for the summary functions AVG() and FREQ() to be applied to them.

But, if you specify

```
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);
```

the query 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 your client session.

You can also use RSPT to join server data with client data. For example, if you specify

```
libname mylib 'c:\sales';

proc sql;
   connect to remote
      (server=tso.shr1 dbms=db2
       dbmsarg=(ssid=db2p));

   select * from mylib.sales97,
      connection to remote
         (select qtr, division,
                 sales, pct
            from revenue.all97
            where region='Southeast')
      where sales97.div=division;
```

the subquery against the DB2 data is sent through the SAS server to the DB2 server, and 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.SALES97.

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, the following 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;
```

CHAPTER 15
Examples Using Remote SQL Pass-Through (RSPT)

Example 1. RSPT Services: Querying a Table in DB2

Purpose

The following example shows how to query a DB2 table that is located on a remote machine by using SQL statements issued from a client session.

Program

The following sequence of statements would be 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');
```

Use the AS alias clause in the CONNECT TO statement to give the same name to the connection to the remote DBMS as it would have if you connected directly to it. This 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: Subsetting Remote SAS Data

Purpose

The PROC SQL view SALES03 presents sales data for fiscal year 2003 and is defined on a UNIX machine as follows:

```
create view servlib.sales03 as
   select customer, sum(amount) as amount
   from sales
   where year=2003 and
      salesrep='L. Peterson'
      group by customer
      order by customer;
```

Processing this view (by using RLS from a Windows client session) is relatively fast because the view is interpreted in the server session. The summary function SUM() is applied when the view is interpreted and only the summary row is returned to the client session.

RLS Program

You can create a new view in your local SAS library to access the underlying data by using RLS from your Windows client session, as follows:

```
libname mylib 'C:\sales';

libname servlib '/dept/sales/revenue'
   server=servername;

create view mylib.sales03 as
   select customer, sum(amount) as amount
      from servlib.sales
      where year=2003 and
         salesrep='L. PETERSON'
         group by customer
         order by customer;
```

However, processing this view is expensive because the summary is not performed until the data reaches the client session. This means more data is sent across the network. In the following RSPT example, the summary is done before the data is transferred. This reduces the amount of data that crosses the network.

RSPT Program

The following statements create a new PROC SQL view in a local SAS library that uses RSPT to access the remote SAS data:

```
libname mylib 'C:\sales';

libname servlib '/dept/sales/revenue'
   server=servername;

proc sql;
connect to remote
   (server=servername);

create view mylib.sales03 as
   select * from connection to remote
      (select customer, sum(amount) as amount
      from servlib.sales
      where year=2003 and
            salesrep='L. PETERSON'
      group by customer
      order by customer);
```

Note: The libref SERVLIB must 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 remote submit a LIBNAME statement to define the library. △

You might want to create a view in the server session, which 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. The following 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.cust03 as
      select customer,
      sum(amount) as amount from sales
      where year=2003
      group by customer) by remote;
```

CHAPTER 16 Examples of Combining Compute Services and Data Transfer Services

Advantages of Combining Compute Services and Data Transfer Services

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 amount of data is too large.
□ the data is frequently updated.
□ data duplication is to be avoided.

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, prior to 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.

Example 1. Compute Services and Data Transfer Services Combined: Processing in the Client and Server Sessions

Purpose

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.

Program

Suppose that you want to perform some processing on a remote computer, download the resulting SAS data set, create a permanent data set in the client session, and print a report in the client session.

The following example accomplishes these tasks in a single program.

```
     /*************************************/
     /* prepare to sign on                */
     /*************************************/
❶ options
     comamid=tcp
     remote=netpc;
❷ libname lhost 'c:\sales\reg1';

     /*************************************/
     /* sign on and download data set     */
     /*************************************/
❸ signon;
❹ rsubmit;
❺    libname rhost 'd:\dept12';
❻      proc sort data=rhost.master
        out=rhost.sales;
        where gross > 5000;
        by lastname dept;
     run;

❼    proc download data=rhost.sales
        out=lhost.sales;
     run;
❽endrsubmit;


❾      /*************************************/
     /* print data set in client session  */
     /*************************************/
 proc print data=lhost.sales;
 run;
```

❶ Specify the COMAMID= and the REMOTE= system options in an OPTIONS statement. These two system options define the connection between the client and server sessions.

❷ Define a libref for the SAS data library in the client session to identify the location of the data set that was downloaded.

❸ Sign on to the server session. The *server-ID* was specified in the preceding OPTIONS statement.

Note: A script file is not used. △

❹Use 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.

❺ Define the libref for the SAS data library in the server session.

❻ Create the RHOST.SALES data set as a sorted subset of the RHOST.MASTER data set.

❼ The PROC DOWNLOAD step transfers the SALES data from the library in the server session (RHOST) to the library in the client session (LHOST).

❽ The ENDRSUBMIT statement signals 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.

❾ The PROC PRINT step executes in the client session and reads the SAS data set that was downloaded in the PROC DOWNLOAD step.

Running the Program

You have several choices for running this program:

- □ Type and submit each line in a 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 a 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 “Using SAS Explorer to Monitor SAS/CONNECT Tasks” on page 114.

Example 2. Compute Services and Data Transfer Services Combined: Sorting and Merging Data

Purpose

In cases where the same remote data set needs to be manipulated in multiple client sessions, data transfer 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 its Compute Services to process that data in the client session. With this method, client sessions do not have to continually access the data set on the remote machine.

Program

The following SCL program fragment distributes a reservations data set from a remote computer 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.

```
  INIT:
  submit continue;
  signon atlanta;

rsubmit;
     libname mres "d:\counter";
     libname backup "d:\counter\backup";

❶     proc upload data=mres.reserv
         out=combine status=no;
         where origin="Atlanta";
      run;

❷     proc sort data=combine;
         by resnum;
      run;

❸     proc copy in=mres out=backup;
         select reserv;
      run;

❹     data mres.reserv;
         update mres.reserv combine;
         by resnum;
      run;
  endrsubmit;

  signoff;
```

❶ Upload all reservations for a particular location.

❷ Sort uploaded data sets for merging.

❸ Back up existing data set.

❹ Merge new and existing data sets.

Example 3. Compute Services and Data Transfer Services Combined: Macro Capabilities

Purpose

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 non-zero 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.

Program

Suppose that you have a transaction file on your client computer and you want to upload it to a remote machine, and then use it 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 if 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.

```
❶ libname trans 'client-SAS-data-library';
  libname backup 'client-SAS-data-library';
❷ rsubmit;
❸    proc upload data=trans.current out=current;
     run;

❹    %sysrput upload_rc=&sysinfo;
     %macro update_employee;

❺       %if &sysinfo=0 %then %do;
           libname perm 'server-SAS-data-library';
           data perm.employee;
              update perm.employee current;
              by employee_id;
           run;
        %end;

❻       %else %put ERROR: UPLOAD of CURRENT
                   failed. Master file was
                   not updated.;
     %mend update_employee;
❼    %update_employee;
  endrsubmit;

❽ %macro check_upload;
❾    %if &upload_rc=0 %then %do;
❿       proc datasets lib=trans;
           copy out=backup;
        run;
     %end;
  %mend check_upload;
⓫ %check_upload;
```

❶ Associate a libref with the SAS data library that contains the transaction data set and backup data in the client session.

❷ Send the PROC UPLOAD statement and the UPDATE_EMPLOYEE macro to the server session for execution.

❸ 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.

❹ 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 if the PROC UPLOAD step was successful.

❺ Test 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.

❻ 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.

❼ Executes the UPDATE_EMPLOYEE macro in the server session.

❽ The CHECK_UPLOAD macro is defined in the client session because it follows the ENDRSUBMIT statement.

❾ Test the value of the UPLOAD_RC macro variable that was created by the %SYSRPUT macro statement in the server session to determine if the PROC UPLOAD step was successful.

❿ 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.

⓫ Execute the CHECK_UPLOAD macro in the client session.

CHAPTER 17

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.

Requestor Window Appears Despite NOTERMINAL Option Setting

Despite your setting the NOTERMINAL option to suppress the display of a requestor window in the server session, a requestor window appears when you use the RSUBMIT statement and the WAIT= option.

To prevent this window from appearing, 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 remote-submit code that uses square brackets because the local machine’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 (]).

For OpenVMS Alpha, square brackets are usually used to delineate the directory name in a pathname. However, you can use < and > as equivalent delimiters. For example:

```
libname sales 'disk:<sales.years.1991>';
```

No Terminal Connected to SAS Session

After remote-submitting a full-screen procedure, you receive the following message:

```
ERROR:  No terminal connected to the SAS session.
```

SAS/CONNECT does not support remote submission of full-screen procedures. You might be able to issue a LIBNAME statement, and use the full-screen 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: Preventing Pipes from Closing Prematurely” on page 162.

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 machine that you are trying to sign on to. For details about starting a spawner, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

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.

P A R T 5

Remote Library Services

CHAPTER 18
Remote Library Services (RLS)

Introduction to Remote Library Services

RLS: 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 machines that have different architectures. RLS also provides read-only access to some SAS catalog entry types across machines that have different architectures.

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

Client Access to a Single- or Multi-User Server

To access a SAS data library on a server that you are already signedon 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 Statement and Command" on page 63.

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 details, see the OPERATE procedure in the *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 Statement and Command" on page 63.

RLS: 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 if

- 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 allows 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). Applications can be built 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.

Use 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.

Use 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.

Use 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.

Use 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.

Compare DTS, RLS, and CS

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. The entire 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.

Using RLS to Access Types of Data

RLS Support for Data Types

RLS supports access to the following types of data:

- SAS catalog*
- SAS file (data set and 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 machines that the client and the server run on do not have compatible architectures.

**Only SAS/CONNECT allows update access to external databases.

Accessing a Catalog

In order for a client to use RLS to update a catalog on a server, the architectures of the machines on which the client and the server run must be compatible. If machine 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.
```

Accessing 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.

Accessing a View

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

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

For all PROC SQL views or for any other type of view processing between a client and a server whose machine architectures are compatible, the view can be interpreted at the client. To interpret a view at the client instead of at the server, set the RMTVIEW= option to NO in a LIBNAME statement. An example follows:

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

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

Accessing 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 machines that the client and the server run on must be compatible. If machine 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 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.

Using Data Views with Servers

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

RLS can be used with three types of views:

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

A view is a SAS file that contains no data, but describes other data. A 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 view is accessed through a server, the 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 view is involved.

If the view is a PROC SQL view or if the client and server machine architectures are the same, you can cause the 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 only be interpreted in the server session.

Interpreting a view as data can produce significant processing demands. When a 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 if

- the number of observations that are produced by the view is much smaller than the number of observations that are read by the view.
- the data sets that are read by the 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 if

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

Using 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 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 Examples 2, 4, and 6 in Chapter 21, "Examples Using Remote Library Services (RLS)," on page 203.

CHAPTER 19 Syntax for the LIBNAME Statement

LIBNAME Statement

Associates a libref (a shortcut name) with a SAS data library that is located on the server for client access.

Valid: Client Session

Category: Data Access

See: LIBNAME Statement in the documentation for your operating environment.

See Also: Base LIBNAME statement

Syntax

LIBNAME *libref* <*engine*> <'*SAS-data-library*'> SERVER=*server-ID* <*options*> <*engine/host-options*>;

Arguments

libref
specifies the name of a library reference to a SAS data library that is located on the server. The libref that you specify is presumed to be the server libref for an existing server library unless you specify 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.

engine
specifies the name of a valid SAS engine for a client to access the server library. Usually, 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*.

engine is a positional argument. If you use it, it must follow the libref.

CAUTION:

Do not confuse the *engine* option with the RENGINE= option. An engine is used by a client to access a server. An RENGINE is used by the server to access its SAS data library. △

'SAS-data-library'

specifies the physical name for the SAS data 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-data-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 data library, see the documentation that is appropriate to your operating environment.

SERVER=*server-ID*

specifies the ID of the server (where the SAS data 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. For details, see "SIGNON Statement and Command" on page 63. To specify a server name that contains more than 8 characters, you must store the name in a macro variable.

Options

ACCESS=READONLY

controls a client's read access to a SAS data library on the server. If you specify this option, you can read but not update data in the 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 data library on the server. SLIBREF=*server-libref* and *'SAS-data-library'* are mutually exclusive.

Engine-Host Options

RENGINE=*engine-name*

specifies the engine for the server session to use to access the SAS data library on the server. Using this option is usually unnecessary because the server automatically determines which engine is used to process 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 which engine to use.

CAUTION:

Do not confuse the RENGINE= option with the *engine* argument. An RENGINE is used by the server to access its SAS data library. An engine 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 will process the SAS data 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 will use. For details about options that are specific to an operating environment, see the documentation that is appropriate for the operating environment used.

RMTVIEW=YES | NO

determines whether SAS data views are interpreted in the server session or the client session. SAS data 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 data views, like SAS data sets, are accessed through an engine. Where a data 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 Institute creates a product-specific engine for each SAS/ACCESS interface product that the SAS/ACCESS views use for that interface.

When views are interpreted in the server session, the server session might require large amounts of processor time and storage, but 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 RMTVIEW to NO causes SAS data views to be interpreted at the client.

Default: YES, which causes views to be interpreted in the server session.

Examples

Example 1: Assigning and Defining a Libref to Access a Library on a Server The following statement associates the libref SQLDSLIB with the SAS data library SASXYZ.VIEWLIB.SASDATA. This library will be accessed through the server MVSHOST, which is running in a server session.

```
libname sqldslib 'sasxyz.viewlib.sasdata' server=mvshost;
```

Example 2: Associating 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: Specifying a Server in the LIBNAME Statement The following example shows a spawner invocation on a machine named MYHOST.MY.NET.WORK. The -SERVICE option specifies that the spawner listens for client connections on port 2323.

```
spawner -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 machine 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 machine 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 machine name and port number */
signon remname;
libname scorcard '.' server=remname;
```

CHAPTER
20

Syntax for the LIBNAME Statement, SASESOCK Engine

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: Client Session and Server Session

Category: Data Access

See: LIBNAME statement in the documentation for your operating environment.

See Also: LIBNAME statement in Base SAS documentation.

Syntax

LIBNAME *libref* SASESOCK "*port-specifier* " <TIMEOUT=*time-in-seconds*>;

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 hard-coded port number that specifies an explicit port on the machine where the asynchronous RSUBMIT is executing.
Example:

```
LIBNAME payroll SASESOCK ":256";
```

"*:port service*"
specifies the name of the port service on the machine where the asynchronous RSUBMIT is executing.

Note: The port service must be configured in the SERVICES file in order to use it. If the port that you specify is in use, access will be denied. △
Example:

```
LIBNAME payroll SASESOCK ":pipe1";
```

"*machine-name:port-number*
specifies an explicit port number on the machine that is specified by *machine-name*.

Note: The port number does not have to be configured in the SERVICES file in order to use it. However, if the port that you specify is in use, access will be denied. △
Example:

```
LIBNAME payroll SASESOCK "apex.finance.com:256";
```

"*machine-name:port service*"
specifies the name of the port service on the machine that is specified by *machine-name*.

Note: The port service must be configured in the SERVICES file in order to use it. △
Example:

```
LIBNAME payroll SASESOCK "apex.finance.com:pipe1";
```

alias-for-automatically-selected-port
causes the first available port to be dynamically assigned to the alias that you specify. Use this method if you do not want to specify an explicit port number or a port service. Without a machine specification, the port is selected from the machine that the asynchronous RSUBMIT is executing on. After a port is assigned to the specified alias, you can refer to that port by using the alias without knowing the identity of that port.

Note: If using an automatically selected port, you do not have to configure a port in the SERVICES file. △

Note: To use this method, the SAS Open Metadata Server must be running. For conceptual information and details about setting up a server, see http://support.sas.com/rnd/eai/openmeta/v9/. For details about SAS system options for configuring metadata (names begin with META), see *SAS Language Reference: Dictionary* △
Example:

```
LIBNAME payroll SASESOCK "mypipe";
```

For details about configuring port services in the SERVICES file, see *Communications Access Methods for SAS/CONNECT and SAS/SHARE*. For an explanation of MP CONNECT with piping, see "Pipeline Parallelism" on page 109. For an example of a SAS/CONNECT application that implements MP CONNECT with piping, see "Example 6: Using MP CONNECT with Piping" on page 161.

TIMEOUT=*time-in-seconds*
specifies the time in seconds that a SAS process will wait to successfully connect to another process.

Example:

```
libname in1 sasesock ":pipe1" timeout=50;
```

Default: 10

CHAPTER 21

Examples Using Remote Library Services (RLS)

Example 1. RLS: Accessing Server Data to Print a List of Reports

Purpose

The following example shows a client using RLS to access a modest amount of data on a server in order to print a list of reports. This is a good use of RLS if the data set REPORTS.REQUEST has a small number of observations.

Program

```
   signon rempc;
1  libname reports REMOTE 'd:\prod\reports' server=rempc;
      data _null_;
      set reports.request;
      if (copy = "Y") then do;
         put "Report " report_name
            " has been requested";
      end;
   run;
```

❶ Define 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. RLS: Accessing 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 the impact on network traffic. The WHERE statement moves to client processing only those observations that a report is being requested for. This move is more efficient than moving every observation to client processing and checking the COPY variable for a **Y** value.

Program

```
   signon rempc;

❶ libname reports 'd:\prod\reports' server=rempc;

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

❶ Define a server library to a client session.

❷ Use the WHERE statement to filter unneeded observations.

Example 3. RLS: Updating Server Data

Purpose

This example enables you to take advantage of a mainframe's superior data handling and security features, while 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

```
signon remos390;

❶ libname rlib REMOTE 'hrs.emp.data' server=remos390;

❷ proc fsedit data=rlib.employee;
run;
```

❶ Define the server session human resource library to the client session.

❷ Execute a client FSEDIT to update the employee data set that is located on the z/OS machine.

Example 4. RLS: 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 in this case would be poor, at best. 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 os390;
libname master REMOTE "hq.prod.data" server=os390;

❶ rdsid = open("master.reserv", 'u');

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

❶ Open the remote Headquarters database.

❷ Build and apply the WHERE clause to accelerate retrieval.

Example 5. RLS: Updating a Server Data Set by Applying a Client Transaction Data Set

Purpose

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

This example creates a data set by remote submitting a DATA step. Next, it creates a client transaction data set. Using RLS, it assigns a client LIBNAME to the server library. Finally, the program modifies the server data set with the client transactions.

Program

```
%let rsession=unxhost;
signon remote=rsession;
   rsubmit;
❶    data sasuser.my_budget;
      length category $ 9;
      input category $ balance;
      format balance dollar10.2;
      datalines;
   utilities   500
   mortgage   8000
   telephone  1000
   food       3000
   run;

   endrsubmit;

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

❸ libname rlslib slibref=sasuser server=rsession;

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

❺ data _null_;
      set rlslib.my_budget;
      put ’Balance for ’ category  @25
```

```
        'is: ' balance;
  run;

❻ signoff;
```

❶ Create the master data set MY_BUDGET in the library SASUSER in the server session.

❷ Create a client or work transaction data set for updating the server data set MY_BUDGET.

❸ Assign a client library to the library SASUSER in the server session.

❹ Apply the transaction data set to the server data set MY_BUDGET.

❺ Review the results. All items except TELEPHONE will be updated.

❻ Sign off the server. The libref RLSLIB is deassigned as part of the sign-off processing.

Example 6. RLS: Subsetting 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 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

```
   ❶ signon srv1;
      libname remlib '/u/user1/reservations' server=srv1;

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

❸ proc summary data=tmp
      vardef=n noprint;
      by company;
      output out=tmp2;
   run;

❹ ods html body="body.htm";

❺ proc report ls=74 ps=85 split=
   "/" 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;
```

❶ Submit the LIBNAME statement in the client session to define the server library.

❷ 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.

❸ Summarize the client data set.

❹ Create an HTML file.

❺ Create a report using the client summary data set.

CHAPTER 22 Example of Combining RLS and Data Transfer Services (DTS)

Introduction

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.

Example 1. RLS and UPLOAD/DOWNLOAD Combined: Distribution of Reports over a Network

Purpose

The following 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 initiate a conversation with each of the franchise workstations, and to transfer a set of reports to the franchise offices based on selection criteria.

Program

```
   /************************************/
   /* Name: DISTREPORT.SCL             */
   /*                                  */
   /* This program distributes reports */
   /* to the franchise offices.        */
   /************************************/
length rc 8;
```

```
  INIT:

  submit continue;
     /************************************/
     /* set up distribution macro        */
     /************************************/
❶ %macro distribution;

❷ %let franchise_city=
     Atlanta NYC LA Dallas Chicago;
  %let franchise_host=
     tsoatl unixnyc unixla wntdal cmshq;

❸ %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);

❹ options remote=&nextrem
     comamid=communication-access-method;
  filename rlink 'script-file-name';
  signon;

❺ x "alloc fi(xferrpt)
     da('sasinfo.sugi18.xferrpt') shr";

❻ rsubmit;
     filename frptlib
        "d:\counter\reports\prod";
  endrsubmit;

     /************************************/
     /* use SAS/CONNECT server           */
     /************************************/
❼ libname rpt "d:\counter\reports" server=&nextrem;
❽ 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              */
        /*********************************/
❾    if (copy="Y") then do;
        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   */
      /**********************************/
⓾ %include xferrpt;

   signoff;
   %end;

   %mend;

      /**********************************/
      /* invoke macro to distribute     */
      /* reports                        */
      /**********************************/
⓫ %distribution;
  endsubmit;

   _status_='H';

   return;

   MAIN:
      return;

   TERM:
      return;
```

❶ Begin the distribution macro definition.

❷ Initialize the list of remote franchise offices (**`franchise_city`**) and their node names (**`franchise_host`**) to be used as the REMOTE= value.

❸ Scan to the next office and node name to be processed.

❹ Specify the remote office NODENAME as the REMOTE= value and sign on to the remote franchise.

❺ Allocate a z/OS file that will contain generated UPLOAD statements.

❻ Remote submit a fileref to define the PC library to which reports will be uploaded.

❼ Connect to a single-user server to access the library that contains the report-selection data set.

❽ Execute the DATA step to evaluate report-selection data (RPT.PREPORT) and create UPLOAD statements to transfer reports (XFERRPT).

❾ If the selection criterion is YES, create the appropriate PROC UPLOAD statement for the specified report.

❿ Include the generated SAS job in the client session for execution.

⓫ Invoke the macro.

PART 6

Data Transfer Services

CHAPTER 23 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 machine architectures (for example, z/OS and Windows) or different SAS software releases (for example, Version 8 and Version 9). Data Transfer Services automatically translates the internal representations of character and numeric data between the client and the server machines.

Note: The translation algorithm was changed between Version 6 and Version 8 and later releases of SAS. See “File Format Translation Algorithms” on page 315. △

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 work load 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 machines 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 facilitate 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 machine 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 method can result in significant savings of server resources and convenience for you.

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 Resource

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 Resource

If the client accesses a small to medium amount of data, Remote Library Services allows the processing to occur at the client, with the 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 examples of combined services, see Chapter 16, "Examples of Combining Compute Services and Data Transfer Services," on page 177 and Chapter 22, "Example of Combining RLS and Data Transfer Services (DTS)," on page 209. To understand these examples, you must be familiar with the syntax for the UPLOAD and DOWNLOAD procedures (described in Chapter 24, "The UPLOAD Procedure," on page 223 and Chapter 25, "The DOWNLOAD Procedure," on page 239).

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 Version 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 details, see the COMPRESS= data set option in *SAS Language Reference: Dictionary*.

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 perfomed:

- □ 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. An example follows:

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

△

Transfer Status Window

The Transfer Status window displays information that describes the status of the download or upload operation. The display of the Transfer Status window is determined by the setting of the CONNECTSTATUS= option, which can be specified in the following contexts:

- □ CONNECTSTATUS= system option. See "CONNECTSTATUS System Option" on page 18.
- □ CONNECTSTATUS= option in the RSUBMIT statement. See "RSUBMIT Statement and Command" on page 131.
- □ CONNECTSTATUS= option in the SIGNON statement. See "SIGNON Statement and Command" on page 63.
- □ CONNECTSTATUS= option in the PROC UPLOAD statement. See Chapter 24, "The UPLOAD Procedure," on page 223.
- □ CONNECTSTATUS= option in the PROC DOWNLOAD statement. See "PROC DOWNLOAD Statement Options" on page 241.

The display on the window changes as the transfer proceeds. The information on the display includes:

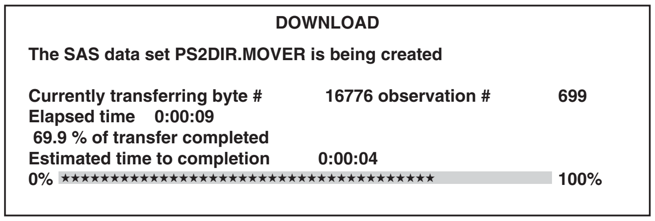
- □ The type of file that is being transferred (SAS data set, SAS catalog, catalog entry that contains graphics output, external file, or SAS utility file).
- □ The name of the target SAS data set, SAS catalog, external file, or SAS utility file. SAS data set names have the form *libref.SAS-data-set*. SAS catalog names have the form *libref.SAS-catalog*. External file names are displayed with the complete filename. Utility file names have the form *libref.SAS-utilityfilename*.
- □ The number of the byte that is being transferred (updated as each new buffer is sent).
- □ The number of the observation that is being transferred (for SAS data sets only).
- □ The time that elapsed since the beginning of the transfer, in *hh:mm:ss* form.
- □ The percentage of the file that is already transferred.
- □ An estimate of the amount of time that is required to complete the transfer, in *hh:mm:ss* form.
- □ A horizontal bar chart that depicts the percentage of the file that is already transferred.

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 PS2DIR.MOVER.

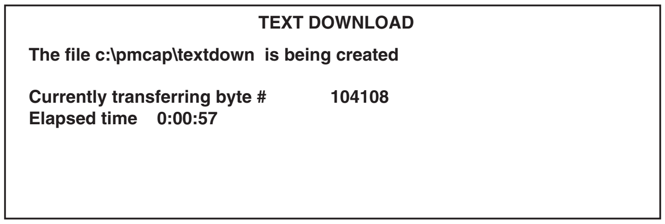
Display 23.1 Transfer Status Window for Downloading a SAS Data Set



```
                         DOWNLOAD
The SAS data set PS2DIR.MOVER is being created

Currently transferring byte #       16776 observation #       699
Elapsed time   0:00:09
 69.9 % of transfer completed
Estimated time to completion       0:00:04
0% *************************************                   100%
```

The following display is an example of the Transfer Status window when an external text file is downloading. The target file is **`C:\PMCAP\TEXTDOWN`**. In this example, because the server is unable to provide the size of the input file, the Transfer Status window omits the percentage of transfer completed, the estimated time to completion, and the bar chart.

Display 23.2 Transfer Status Window for Downloading an External File



```
                       TEXT DOWNLOAD
The file c:\pmcap\textdown  is being created

Currently transferring byte #       104108
Elapsed time   0:00:57
```

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:
  1. the size and number of files that are being transferred
  2. the processing load on the server
  3. the communication access method that is being used
  4. the network configuration.

  The Transfer Status window keeps you informed of the progress of the transfer. For details, see "Transfer Status Window" on page 219.
- □ 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 in the FILENAME statement, see the FILENAME statement in the *SAS Companion for z/OS*.

- 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 successfully completed, 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 Statement” on page 145.
- 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 the FILENAME statement in the *SAS Companion for z/OS*.

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

  ```
  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 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.

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 using 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. The SAS Support Consultant for the server should create the alternate table.
- □ If SAS/CONNECT is not used frequently, you can manage problematic characters by assigning the correct hex 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;
```

CHAPTER 24 The 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

- 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.

The syntax and specifications for the UPLOAD procedure are provided here. For examples that use this syntax, see

Chapter 23, "Using Data Transfer Services," on page 215

Chapter 16, "Examples of Combining Compute Services and Data Transfer Services," on page 177

Chapter 22, "Example of Combining RLS and Data Transfer Services (DTS)," on page 209.

Syntax for the UPLOAD Procedure

PROC UPLOAD
<*data-set-option(s)*>
<*catalog-option(s)*>
<*library-option(s)*>
<*external-file-option(s)*>
<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 | (DISP SRC)>;

Syntax for the PROC UPLOAD Statement

Transfers files from the client to the server.

Valid in: Client Session

Category: Data Access

Syntax

PROC UPLOAD
<*data-set-option(s)*>
<*catalog-option(s)*>
<*library-option(s)*>
<*external-file-option(s)*>
<AFTER=*date*>
<CONNECTSTATUS=YES | NO>;

Syntax Description

- *data-set-options* can be one or more of the following:

  CONSTRAINT=YES | NO

  DATA=*client-SAS-data-set*

  DATECOPY

  EXTENDSN=YES | NO

  INDEX=YES | NO

  OUT=*server-SAS-data-set*

  V6TRANSPORT

- *catalog-options* can be one or more of the following:
  - ENTRYTYPE=*etype*
  - EXTENDSN=YES | NO
  - INCAT=*client-SAS-catalog*
  - OUTCAT=*server-SAS-catalog*
- *library-options* can be one or more of the following:
  - CONSTRAINT=YES | NO
  - EXTENDSN=YES | NO
  - GEN=YES | NO
  - INDEX=YES | NO
  - INLIB=*client-SAS-library*
  - MEMTYPE=(*mtype-list*)
  - OUTLIB=*server-SAS-library*
  - VIEWTODATA
  - V6TRANSPORT
- *external-file-options* are the following:
  - BINARY
  - INFILE=*client-file-identifier*
  - OUTFILE=*server-file-identifier*

PROC UPLOAD Statement Options

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 machines. If a machine 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 Releases 6.09E, 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 Release 6.11 TS020 or Release 6.08 TS415 through Release 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.

For an example of using the BINARY option, see "Example 10. DTS: Distributing an .EXE File from the Server to Multiple Clients" on page 266.

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. For details, see "Transfer Status Window" on page 219.

Alias: CSTATUS=, STATUS=

Default: YES

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.

DATA=*client-SAS-data-set*

specifies a SAS data set that you want 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.

Requirements: If you specify the DATA= option, you must either specify the OUT= option or omit all other output file options.

DATECOPY

For each data set that is transferred, retains the date on which a SAS data set was created and the date on which a SAS data set was last modified.

ENTRYTYPE=*etype*

specifies a catalog entry type to be uploaded. Examples of catalog entry types include DATA and FORMAT.

Alias: ETYPE=, ET=

Requirements: 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 Version 8 or a later release, and the V6TRANSPORT option is specified, the default is to promote the length of a numeric variable whose length is less than 8 bytes. This is consistent with Version 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 Version 6, neither the V6TRANSPORT nor the EXTENDSN= option is supported or recognized.

See "File Format Translation Algorithms" on page 315 for information about translating file formats between a client and server that run on machines whose internal representations are incompatible.

Default: NO

GEN=YES | NO

specifies that data set generations are to be sent during library transfers.

YES
: data set generations are sent during library transfers.

NO
: 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 data 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 data 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, 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 an error message is displayed.

```
WARNING: FILEFMT entries
```

△

INDEX=YES | NO

specifies whether to re-create an index at the server when you upload 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 upload a single data set and omit the OUT= option, or if you upload a SAS data library, the index is re-created by default.

If you specify the OUT= option and the DATA= option, the index is not re-created.

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 which file(s) in that aggregate storage location should be transferred. An asterisk (*) can be used as a wildcard character in the *member* specification of the files to transfer.

- □ All files in the specified location (*).
- □ All files that have the same extension (*.extension).
- □ All files that have the same name but different extensions (name.*).

You must define the fileref before specifying the PROC UPLOAD statement. For details about filerefs for your operating environment, see the appropriate operating environment companion documentation.

The following example shows how to use a wildcard to transfer (upload) all files whose filenames have the extension *.sas* from a client that runs under a Windows operating environment to a directory on a server that runs under the UNIX operating environment.

```
filename locref 'c:\';
rsubmit;
   filename fref '/local/programs';
   proc upload infile=locref('*.sas')
               outfile=fref;
   run;
endrsubmit;
```

'external-file-name'

is used to explicitly define the file that is to be uploaded.

INLIB=*client-SAS-library*

specifies a SAS data 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.

Valid member types are:

- □ ALL
- □ CATALOG
- □ DATA

- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirements: To use this option, you must also specify the INLIB= and OUTLIB= options.

OUT=*server-SAS-data-set*

names the SAS data set on the server 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 Version 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 Version 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 how to interpret 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 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 OUT= 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 data 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 231.

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 specify a two-level SAS catalog name. To upload all of the catalogs in a SAS data library, specify

OUTCAT=*libref*._ALL_

Requirements: 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.

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 remote submit) the following statements to the SAS System:

```
proc build cat=libref.member-name batch;
   compile;
run;
```

where *libref* identifies the SAS data library that contains the catalog, and *member-name* identifies the catalog.

OUTFILE=*server-file-identifier*

identifies an external file on the server that you want to upload the external file to. *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 UPLOAD 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 which file in that aggregate storage location should be transferred. You must define the fileref before specifying the PROC UPLOAD 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 uploaded.

Requirements: If you use the OUTFILE= option, you must also use the INFILE= option.

OUTLIB=*server-SAS-library*

names the destination SAS data 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 Version 6 data translation algorithm. Specify this option only when you want to use the Version 6 translation style explicitly and both the client and the server run Version 8 or a later release of SAS. For details about the data transfer algorithms, see "File Format Translation Algorithms" on page 315.

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.

Default Naming Conventions for Uploaded Data Sets

If you omit the OUT= option, which specifies the name of the output data set, 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 environment, 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-data-library;
  ```

  If you remote 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-data-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 environment, the data set is uploaded to the default library on the server. This is usually the WORK library, but the library may 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 remote 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-data-library;
  options user=orders;
  ```

  If you remote 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-data-library;
libname remote
   server-SAS-data-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, 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-data-library;
options user=orders;
```

  When you remote 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-data-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.

Attributes and Data Set Options for the DATA= and OUT= Options

PROC UPLOAD permits you to specify SAS data set options in the DATA= and OUT= options.

Note: SAS data set options are not supported with 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. Therefore, the data set options must be used in the DATA= or OUT= options. △

In addition, when you upload SAS data sets by using the DATA= option (omitting the OUT= option) or by using the INLIB= and OUTLIB= options, or if you omit all of these options, the following characteristics are inherited by the uploaded data set.

Note: The following list of characteristics shows the SAS data set option (in parenthesis) that was used to create the characteristic for the original data set. △

- □ password for ALTER protection (ALTER= SAS data set option)
- □ compressed observations (COMPRESS= SAS data set option)
- □ indexes (INDEX= SAS system option or other methods of creating indexes).

  Note: The index for an uploaded SAS data set is re-created on the server; not copied from the client. To prevent the re-creation of the index, you can specify the INDEX=NO option in the PROC UPLOAD statement, as described in "PROC UPLOAD Statement Options" on page 225. △
- □ data set label (LABEL= SAS data set option)
- □ password for READ protection (READ= SAS data set option)
- □ re-use of free space in compressed data sets (REUSE= SAS data set option)
- □ list of variables that the data set is sorted by (SORTEDBY= SAS data set option)
- □ data set type (TYPE= SAS data set option)
- □ password for WRITE protection (WRITE= SAS data set option)
- □ integrity constraints
- □ maximum generations (GENMAX= SAS data set option).

If you specify the OUT= option when uploading a single data set, only the following characteristics are inherited by the uploaded data set:

- □ data set type
- □ data set label.

The following example illustrates the use of the DATA= option and the INDEX=NO option. The example also shows the use of the KEEP= SAS data set option. Notice that because no OUT= option is specified, the uploaded data set inherits the characteristics of the input data set except the index (because the INDEX=NO option is specified).

```
proc upload data=study(keep=age score1 score2) index=no;
run;
```

Syntax for the WHERE Statement

Selects observations from SAS data sets.

Restrictions: The UPLOAD procedure processes WHERE statements when you transfer a single SAS data set.

See also: WHERE Statement Syntax in *SAS Language Reference: Dictionary*.

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 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 WHERE statement in the *SAS Language Reference: Dictionary*.

Syntax for the EXCLUDE Statement

Excludes library members or catalog entries from uploading.

Restrictions: 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.

Restrictions: *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.
Valid member types are:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirements: 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=

Requirements: To use this option, you must specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

Syntax for the SELECT Statement

Selects specific library members or catalog entries to upload.

Restrictions: 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 upload. 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 uploading.

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 uploading.

Restrictions: *first* and *last* must begin with identical character strings and must end in a number.

cat-entry-list
specifies which catalog entries to upload. Each element of *cat-entry-list* has the form *entry.type*.

entry	is the name of an entry in the *catalog* to upload.
.type	is the type of the catalog entry. This part of the name is optional.

MEMTYPE=*mtype*
specifies a member type to upload.
Valid member types are:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirements: 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 upload. Examples of catalog entry types include FORMAT and DATA.

Alias: ETYPE=, ET=

Requirements: To use this option, you must specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

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 in the order that you specify them in the SELECT statement. △

Syntax for the 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

Requirements: To use the TRANTAB statement, you must specify the INCAT= and OUTCAT= options in the PROC UPLOAD statement.

Restrictions: You can only specify one translation table per TRANTAB statement. To specify additional translation tables, use additional TRANTAB statements.

See: The TRANTAB Statement for the SAS/CONNECT UPLOAD and DOWNLOAD procedure in the *SAS National Language Support (NLS): User's Guide*

TRANTAB NAME=*translation-table-name*
<*option(s)*>;

PROC UPLOAD Output

The UPLOAD 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 24.1 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.
$
```

CHAPTER 25
The DOWNLOAD Procedure

Introduction

After you have started SAS/CONNECT, you can transfer files between your client session and the server. The DOWNLOAD procedure copies files that are stored on the server to the client.

Using PROC DOWNLOAD, you can

- □ 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 which translation table should be used when you download a SAS catalog.

The syntax and specifications for the DOWNLOAD procedure are described here. For examples that use this syntax, see

Chapter 23, “Using Data Transfer Services,” on page 215

Chapter 16, “Examples of Combining Compute Services and Data Transfer Services,” on page 177

Chapter 22, “Example of Combining RLS and Data Transfer Services (DTS),” on page 209.

Syntax for the DOWNLOAD Procedure

PROC DOWNLOAD
  <*data-set-option(s)*>
    <*catalog-option(s)*>
    <*external-file-option(s)*>
    <*library-option(s)*>
    <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 | (DISP SRC)>;

Syntax for the PROC DOWNLOAD Statement

Transfers files from the server to the client.

Valid in: Client Session

Category: Data Access

Syntax

PROC DOWNLOAD
  <*data-set-option(s)*>
    <*catalog-option(s)*>
    <*library-option(s)*>
    <*external-file-option(s)*>
    <AFTER=*date*>
    <CONNECTSTATUS=YES | NO>;

Syntax Description

- *data-set-options* can be one or more of the following:

  CONSTRAINT=YES | NO

  DATA=*server-SAS-data-set*

  DATECOPY

  EXTENDSN=YES | NO

  INDEX=YES | NO

  OUT=*client-SAS-data-set*

  V6TRANSPORT

- □ *catalog-options* can be one or more of the following:
  - ENTRYTYPE=*etype*
  - EXTENDSN=YES | NO
  - INCAT=*server-SAS-catalog*
  - OUTCAT=*client-SAS-catalog*
- □ *library-options* can be one or more of the following:
  - CONSTRAINT=YES | NO
  - EXTENDSN=YES | NO
  - GEN=YES | NO
  - INDEX=YES | NO
  - INLIB=*server-SAS-library*
  - MEMTYPE=(*mtype-list*)
  - OUTLIB=*client-SAS-library*
  - VIEWTODATA
  - V6TRANSPORT
- □ *external-file-options* are the following:
  - BINARY
  - INFILE=*server-file-identifier*
  - OUTFILE=*client-file-identifier*

PROC DOWNLOAD Statement Options

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 machines. If a machine 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 Releases 6.09E, 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, 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), 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 10. DTS: Distributing an .EXE File from the Server to Multiple Clients" on page 266.

CONNECTSTATUS=YES | NO

specifies whether the status window should be displayed during a transfer. By default, the DOWNLOAD procedure displays the Transfer Status Window (CONNECTSTATUS=YES). For details, see "Transfer Status Window" on page 219.

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*

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.

Requirements: If you specify the DATA= option, you must either use the OUT= option or omit all other options.

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=

Requirements: 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 machine.

The behavior of the EXTENDSN= option varies according to the SAS release that is used.

- □ If both the client and the server run Version 8 or a later release, and the V6TRANSPORT option is specified, the default is to promote the length of the numeric variable whose length is less than 8 bytes. This is consistent with Version 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 Version 6, neither the V6TRANSPORT nor the EXTENDSN= option is supported or recognized.
- □ If the client runs Version 6 and the server runs Version 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 Version 6 default and to prevent the promotion.

See “File Format Translation Algorithms” on page 315 for information about translating file formats between a client and server that run on machines whose internal representations are incompatible with the other.

Default: NO

GEN=YES | NO
specifies that data set generations are to be sent during library transfers.

YES
data set generations are sent during library transfers.

NO
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 data 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 data 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 and they run different SAS releases, 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 an 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 data library, the index is re-created by default.

If you specify the OUT= option and the DATA= option, the index is not re-created.

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 which file(s) in that aggregate storage location should be transferred. An asterisk (*) can be used as a wildcard character in the *member* specification of the files to transfer.

- ☐ All files in the specified location (*).
- ☐ All files that have the same extension (*.extension).
- ☐ All files that have the same name but different extensions (name.*).

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.

The following example shows how to use a wildcard to transfer (download) all files whose filenames have the extension *.sas* and are located in a directory on a server that runs under a UNIX operating environment to a folder on a client that runs under the Windows operating environment.

```
filename locref 'c:\';
rsubmit;
   filename fref '/local/programs';
   proc download infile=fref('*.sas')
                 outfile=locref;
   run;
```

```
endrsubmit;
```

'external-file-name'
is used to explicitly define the file that is to be downloaded.

INLIB=*server-SAS-library*
specifies a SAS data 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.
Valid member types are:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirements: To use this option, you must also specify the INLIB= and OUTLIB= options.

OUT=*client-SAS-data-set*
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 how to interpret 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 data 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 “Default Naming Conventions for Downloaded Data Sets” on page 247

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 data library, specify

OUTCAT=*libref*._ALL_

Requirements: 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.

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 remote submit) the following statements to the SAS System:

```
proc build cat=libref.member-name batch;
   compile;
run;
```

where *libref* identifies the SAS data library that contains the catalog and *member-name* identifies the catalog.

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.

Requirements: If you use the OUTFILE= option, you must also use the INFILE= option.

OUTLIB=*client-SAS-library*
names the destination SAS data 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 Version 6 data translation algorithms. Specify this option only when you want to use the Version 6 translation style explicitly and both the client and the server run Version 8 or a later release of SAS. For details about the data transfer algorithms, see "File Format Translation Algorithms" on page 315.

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.

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, 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-data-library;
```

  If you remote 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-data-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, the data set is downloaded to the default library on the client. This is usually the WORK library, but the library may 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 remote 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-data-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-data-library;
libname local
   client-SAS-data-library;
   /**********************************/
   /* This option has no effect in     */
   /* this case.                       */
   /**********************************/
options user=local;
```

  If you remote 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-data-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, 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-data-library;
options user=local;
```

  When you remote 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-data-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.

Attributes and Data Set Options for the DATA= and OUT= Options

PROC DOWNLOAD permits you to specify SAS data set options in the DATA= and OUT= options.

Note: SAS data set options are not supported with the INLIB= and OUTLIB= options, even when you download only data sets. The data set options must be associated with a specific SAS data set. Therefore, the data set options must be used in the DATA= or OUT= options. △

In addition, when you download SAS data sets by using the DATA= option (omitting the OUT= option) or by using the INLIB= and OUTLIB= options, or if you omit all of these options, the following characteristics are inherited by the downloaded data set.

Note: The following list of characteristics shows the SAS data set option (in parenthesis) that was used to create the characteristic for the original data set. △

- password for ALTER protection (ALTER= SAS data set option)
- compressed observations (COMPRESS= SAS data set option).
- indexes (INDEX= SAS data set option or other methods of creating indexes).

  Note: The index for a downloaded SAS data set is re-created on the client; it is not copied from the server. To prevent the re-creation of the index, you can specify the INDEX=NO option in the PROC DOWNLOAD statement, as described in "Syntax for the PROC DOWNLOAD Statement" on page 240. △
- data set label (LABEL= SAS data set option)
- password for READ protection (READ= SAS data set option)
- re-use of free space in compressed data sets (REUSE= SAS data set option)
- list of variables that the data set is sorted by (SORTEDBY= SAS data set option)
- data set type (TYPE= SAS data set option)
- password for WRITE protection (WRITE= SAS data set option)
- integrity constraints
- maximum generations (GENMAX= SAS data set option).

If you specify the OUT= option when downloading a single data set, only the following characteristics are inherited by the downloaded data set:

- data set type
- data set label.

The following example illustrates the use of the DATA= option and the INDEX=NO option. The example also shows the use of the KEEP= SAS data set option. Note that because no OUT= option is specified, the downloaded data set inherits the characteristics of the input data set except the index (because the INDEX=NO option is specified).

```
proc download data=study(keep=age score1 score2) index=no;
run;
```

Syntax for the WHERE Statement

Selects observations from SAS data sets.

Restrictions: The DOWNLOAD procedure processes WHERE statements when you transfer a single SAS data set.

See also: WHERE Statement Syntax in *SAS Language Reference: Dictionary*.

Syntax

WHERE *where-expression-1 <logical-operator where-expression-n>*;

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 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 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 the WHERE statement in *SAS Language Reference: Dictionary*.

Syntax for the 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.

Restrictions: *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.
Valid member types are:

- ALL
- CATALOG
- DATA

- ☐ MDDB
- ☐ VIEW

Alias: MTYPE=, MT=

Requirements: 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=

Requirements: To use this option, you must specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

Syntax for the SELECT Statement

Selects specific library members or catalog entries to download.

Restrictions: You cannot use both the EXCLUDE and SELECT statements in the same PROC DOWNLOAD 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 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.

Restrictions: *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.
Valid member types are:

- ALL
- CATALOG
- DATA
- MDDB
- VIEW

Alias: MTYPE=, MT=

Requirements: 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.

Alias: ETYPE=, ET=

Requirements: To use this option, you must specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

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 that you specify them in the SELECT statement.

For an example of using the SELECT statement to maintain the order and grouping of catalog entries that contain graphics output, see "Example 3.4: Using the ENTRYTYPE= Option in Two SELECT Statements in PROC DOWNLOAD" on page 258. △

Syntax for the TRANTAB Statement

Specifies the translation table to use when translating character data for a download from the server to the client

Requirements: To use the TRANTAB statement, you must specify the INCAT= and OUTCAT= options in the PROC DOWNLOAD statement.

Restrictions: You can specify only one translation table per TRANTAB statement. To specify additional translation tables, use additional TRANTAB statements.

See: The TRANTAB Statement for the SAS/CONNECT UPLOAD and DOWNLOAD procedures in the *SAS National Language Support (NLS): User's Guide.*

TRANTAB NAME=*translation-table-name*
    <*option(s)*>;

PROC DOWNLOAD Output

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 25.1 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.
$
```

CHAPTER

26

Examples of Data Transfer Services (DTS)

Example 1. DTS: Transferring Data by Using WHERE Statements

Purpose

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.

Program

```
proc upload data=school out=kindergarten;
   where class='K';
run;
```

Example 2. DTS: Transferring Specific Member Types

Purpose

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 parenthesis.

Programs

Example 2.1: Using the MEMTYPE= Option in the PROC UPLOAD Statement

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.

```
proc upload inlib=this outlib=that
   memtype=(data catalog);
```

Example 2.2: Using the MEMTYPE= Option in the EXCLUDE Statement

This example uploads all catalogs and data sets except the data sets that are named Z4, Z5, Z6, and Z7 that are in the library LOCLIB on the client and stores them in the library REMLIB on the server:

```
proc upload inlib=loclib outlib=remlib mt=all;
   exclude z4-z7 / memtype=data;
run;
```

Example 2.3: Using 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:

```
proc download inlib=remlib outlib=loclib;
   select names salary media(mt=data) / memtype=cat;
run;
```

Example 3. DTS: Transferring Specific Catalog Entry Types

Purpose

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.

If you omit the ENTRYTYPE= option and also omit the SELECT and EXCLUDE statements, all catalog entries are transferred.

Programs

Example 3.1: Using the ENTRYTYPE= Option in the PROC UPLOAD Statement

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:

```
proc upload incat=loclib.cat
   outcat=remlib.upcat entrytype=slist;
run;
```

Example 3.2: Using 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 and stores them in the catalog LOCAL.SECONDARY_FORMATS on the client:

```
proc download incat=remote.main_formats
   outcat=local.secondary_formats;
   exclude xyz grades / entrytype=format;
run;
```

Example 3.3: Using 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 and stores them in the WORK.REMFMT catalog on the server:

```
proc upload incat=locfmt outcat=remfmt;
   select xyz.format grades
      abc (et=format) / et=infmt;
   select a b / et=scl;
run;
```

Example 3.4: Using 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.
Remote 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:

```
proc download incat=rhost.finance
   outcat=lhost.finance;
   select income expense et=grseg;
   select group1.grseg;
run;
```

Example 3.5: Using 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 4. DTS: Transferring Generations of SAS Data Sets

Purpose

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 the user does not want all generations to be transferred, you should either

- □ transfer a library using the GEN=NO option.
- □ transfer single data sets. Only the specified data set is transferred.

Programs

Example 4.1: Using LIBRARY Transfers to Transfer Data Set Generations

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 will be 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 4.2: Using 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 will be 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 will be excluded from the transfer.

In the following example, the data set LOCAL.SALES as well as all of its generations will be uploaded.

```
rsubmit;
   proc upload in=local out=remote cstatus=no;
      select sales (mt=data);
   run;
endrsubmit;
```

Example 4.3: Inheriting 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 Version 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 only inherited 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;
endrsubmit;
```

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.

```
rsubmit;
   proc download data=remote.sales;
   run;
endrsubmit;
```

Example 4.4: Transferring Single Data Sets

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 5. DTS: Transferring Long Member Names

Purpose

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.

Program

```
rsubmit;
   proc upload in=work out=sasuser;
      select longdatasetname(mt=data)
      cat longcatalogname/mt=cat;
   run;

   data x.sas_institute_employee_data;
     set empdata;
   run;

   proc download inlib=x outlib=work;
   run;
endrsubmit;
```

Example 6. DTS: Transferring Data by Using Data Set Options and Attributes

Purpose

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 Chapter 24, “The UPLOAD Procedure,” on page 223 and Chapter 25, “The DOWNLOAD Procedure,” on page 239.

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.

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). △

Program

```
proc download data=survey
   (rename=(r=response) drop=comments)
   index=no;
run;
```

Example 7. DTS: Transferring 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 “Syntax for the PROC UPLOAD Statement” on page 224; for details about PROC DOWNLOAD, see Chapter 25, “The DOWNLOAD Procedure,” on page 239.

Purpose

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 will not be inherited. Also, referential integrity constraint types are not transferred when the referential constraints reside in a different library.

Programs

Example 7.1: Omitting the OUT= Option from the PROC DOWNLOAD Statement

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.

```
proc download data=rem;
run;
```

Example 7.2: Using 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.

```
proc upload data=loc(drop=one);
run;
```

Example 7.3: Using 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.

```
proc upload inlib=sasuser outlib=work;
run;
```

Example 7.4: Using 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, then no integrity constraints are defined for the output data set.

```
proc download data=students index=no;
run;
```

Example 8. DTS: Transferring Numerics by Using the EXTENDSN= and V6TRANSPORT Options

Purpose

For SAS releases prior to Version 8, when transferring short numerics (length less than 8) the length of these numerics is automatically increased to preserve precision. In Version 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 or not to promote the length of numerics.

Programs

Example 8.1: Using the EXTENDSN= and V6TRANSPORT Options in the PROC UPLOAD Statement

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.

```
proc upload data=a out=remote
   v6transport extendsn=no;
run;
```

Example 8.2: Using 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. △

```
proc download incat=remote.scat outcat=work.scat
   extendsn=no;
run;
```

Example 9. DTS: Transferring SAS Utility Files

Purpose

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.

Note: 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
- 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.
Valid values for the MEMTYPE= option are:

- DATA (SAS data sets)
- CATALOG (SAS catalogs)
- VIEW (SQL views)
- MDDB (MDDB files)
- ALL (all of the preceding values).

Programs

Example 9.1: Using the INLIB= Option in the PROC DOWNLOAD Statement

This example downloads 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:

```
proc download inlib=work outlib=work;
run;
```

Example 9.2: Using 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:

```
proc upload inlib=this outlib=that
   memtype=(mddb view);
run;
```

Example 9.3: Using the MEMTYPE= Option in the SELECT Statement

This example downloads 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:

```
proc download inlib=work outlib=local;
   select test1 test2 test3(mt=data)/memtype=mddb;
run;
```

Example 9.4: Using 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 and stores them in the library REMOTE on the server:

```
proc upload inlib=local outlib=remote memtype=all;
   exclude a1-a3/memtype=view;
run;
```

Example 10. DTS: Distributing an .EXE File from the Server to Multiple Clients

Purpose

SAS/CONNECT facilitates the distribution of information to multiple clients. Rather than distributing files on diskettes, 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 machines in your organization. You decide that the most efficient way to update all machines 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 allows all clients to quickly access the updated software, and eliminates the need to share a physical diskette 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. △

Programs

Example 10.1: UPLOAD

The PROGRAM.DLL module must first be uploaded to an external file on the server.

```
rsubmit;
   filename rfile 'server-file';
   proc upload infile='a:\program.dll'
      outfile=rfile binary;
   run;
endrsubmit;
```

This example uses a SAS FILENAME statement to identify the target file on the server.

Note: 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. △

Example 10.2: 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.

```
rsubmit;
   filename rfile 'server-file';
   proc download infile=rfile
      outfile='\sas\sasexe\program.dll' binary;
   run;
endrsubmit;
```

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.

Example 11. DTS: Downloading a Partitioned Data Set from z/OS

Purpose

This example shows how to download all members of a partitioned data set. Suppose you need to download a collection of SAS programs from a z/OS server to your client. The SAS programs are members of one partitioned data set named MFHOST.SAS.PROGRAMS. You can copy all the programs from the partitioned data set to the client by using a single DOWNLOAD procedure. An asterisk (*) wildcard character is specified in the DOWNLOAD procedure to transfer all members of the data set.

Program

```
filename locdir
     '/unixhost/sas/programs';
rsubmit;
   filename inpds
        'mfhost.sas.programs' shr;
   proc download infile=inpds('*')
         outfile=locdir;
endrsubmit;
```

The first FILENAME statement defines the fileref LOCDIR, which identifies the physical location for the files that are downloaded to the UNIX client. The RSUBMIT statement indicates that the statement that follows will be processed on the z/OS server. By not specifying a *server-ID*, this example assumes that the z/OS machine is your current server. The second FILENAME statement defines the fileref INPDS for the partitioned data set MFHOST.SAS.PROGRAMS, which contains the SAS programs that will be downloaded to the client. The PROC DOWNLOAD step transfers all the files in the partitioned data set on the z/OS server to the library LOCDIR on the UNIX client. The ENDRSUBMIT statement indicates the end of the block of statements that are submitted to the server for processing.

Example 12. DTS: Combining Data from Multiple Server Sessions

Purpose

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.

Program

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 ❷ through ❺ apply to downloading both DB2 and Oracle data. △

```
     /************************************/
     /* connect to z/OS                  */
     /************************************/
❶ options comamid=tcp;
  filename rlink
     '!sasroot\connect\saslink\tcptso.scr';
  signon os390host;
     /************************************/
     /* download DB2 data views using    */
     /* SAS/ACCESS engine                */
     /************************************/
❷ rsubmit os390host;
❸ libname db db2;
❹ proc download data=db.employee
     out=db2dat;
  run;
❺ endrsubmit;

     /************************************/
     /* connect to UNIX                  */
     /************************************/
❻ options
     remote=hrunix comamid=tcp;
        filename rlink
           '!sasroot\connect\saslink\tcpunix.scr';
        signon;

     /************************************/
     /* download Oracle data using       */
     /* SAS/ACCESS engine                */
     /************************************/
❷ rsubmit hrunix;
❸    libname oracle user=scott password=tiger;
❹ proc download
     data=oracle.employee out=oracdat;

  run;
❺ endrsubmit;

     /************************************/
     /* sign off both links              */
     /************************************/
❼ signoff hrunix;
  signoff os390host cscript=
     '!sasroot\connect\saslink\tcptso.scr';
```

```
      /*************************************/
      /* join data into SAS view           */
      /*************************************/
❽ proc sql;
   create view joindat as
      select * from db2dat, oracdat
      where oracdat.emp=db2dat.emp;

      /*************************************/
      /* create summary table              */
      /*************************************/
❾ 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 */
❿ 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;
```

❶ 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.

❷ After you connect to two or more sessions, you can remote 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.

❸ Associate a libref with the library that contains the DB2 database on the server.

❹ 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.

❺ The ENDRSUBMIT statement ends the block of statements that are submitted to the server.

❻ 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.

❼ 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.

❽ 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.

❾ To analyze the joined data, use the name of the view on the client in a PROC TABULATE step.

❿ 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.

CHAPTER 27

Data Transfer Services Troubleshooting

Troubleshooting the UPLOAD and DOWNLOAD Procedures

Symbol 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 with single quotation marks, as shown in the following example:

```
PROC UPLOAD INFILE=pcflref
   OUTFILE=hstflref('$filname');
run;
```

Variable-Block Binary File Has LRECL Value Exceeds 256 Bytes

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.

Fixed-Block Binary File Has LRECL Value Exceeds 256 Bytes

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: The RECFM= and LRECL= options in the FILENAME statement are supported only under z/OS operating environments. For details, see the FILENAME statement in the *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 the FILECC= system option in the *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 Version 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 7

Security

CHAPTER 28 Data Security

Security Definitions

Cryptography

Cryptography is an area of research that uses mathematics to provide confidentiality for selective information and to create a high level of trust in sensitive data.

Encryption

Encryption is the transformation of intelligible data (plaintext) into an unintelligible form (ciphertext) by means of a mathematical process. The ciphertext is translated back to plaintext when the appropriate key that is necessary for decrypting (unlocking) the ciphertext is applied.

Authentication

Authentication is the act of verifying the identity of an entity. Usually, authentication is used to confirm the authority of an entity to access protected resources.

Encryption Services, Algorithms, and Comparisons of Services

Encryption Services

The following encryption services are supported:

SASProprietary
: is a fixed encoding algorithm that is included with Base SAS software and is supported under the OpenVMS Alpha, z/OS, UNIX, and Windows operating environments. It requires no additional SAS product licenses. The SAS proprietary algorithm is strong enough to protect your data from casual viewing. However, because a determined hacker can breach this encoding, SAS cannot guarantee that SASProprietary will prevent unauthorized access to your data.

SAS/SECURE
: is an add-on product that provides encryption algorithms in addition to the SAS proprietary algorithm. SAS/SECURE software requires a license, and it must be installed on each machine that runs a client and a server that will use the encryption algorithms. Although SAS/SECURE increases data security, it cannot completely prevent unauthorized access to your data.

SSL
: *Note:* SSL is supported only in the UNIX and Windows operating environments. △

is an abbreviation for Secure Sockets Layer, which is a protocol that provides network security and privacy. Developed by Netscape Communications, SSL uses encryption algorithms that include RC2, RC4, DES, TripleDES, IDEA, MD5, and others. In addition to providing encryption services, SSL performs client and server authentication, and it uses message authentication codes to ensure data integrity. SSL is supported by both Netscape Navigator and Internet Explorer. Many Web sites use the protocol to protect confidential user information, such as credit card numbers. By convention, URLs that require an SSL connection begin with https: instead of http:. The SSL protocol is application independent and allows protocols such as HTTP, FTP, and Telnet to be transparently layered above it. SSL is optimized for HTTP. SSL includes software that was developed by the OpenSSL Project for use in the OpenSSL Toolkit. For more information see **`http://www.OpenSSL.org/`**.

Data Encryption Algorithms

The following encryption algorithms are supported:

RC2
: is a block cipher that encrypts data in blocks of 64 bits. A *block cipher* is an encryption algorithm that breaks down a message into blocks and encrypts each block. The RC2 key size ranges from 8 to 256 bits. In SAS/SECURE, a configurable key size of 40 or 128 bits is used. (The NETENCRYPTKEYLEN= system option is used to configure the key length.) The RC2 algorithm expands a single message to a maximum of 8 bytes. RC2 is a proprietary algorithm developed by RSA Data Security, Inc.

Note: RC2 is supported in SAS/SECURE and SSL. △

RC4
: is a stream cipher. A *stream cipher* is an encryption algorithm that encrypts data 1 byte at a time. The RC4 key size ranges from 8 to 2048 bits. The SAS/SECURE implementation uses a configurable key size of 40 or 128 bits. (The NETENCRYPTKEYLEN= system option is used to configure the key length.) RC4 is a proprietary algorithm developed by RSA Data Security, Inc.

Note: RC4 is supported in SAS/SECURE and SSL. △

DES (Data Encryption Standard)
: is a block cipher that encrypts data in blocks of 64 bits by using a 56-bit key. The algorithm expands a single message to a maximum of 8 bytes. DES was originally developed by IBM but is now published as a U.S. Government Federal Information Processing Standard (FIPS 46-3).

Note: DES is supported in SAS/SECURE and SSL. △

TripleDES
: is a block cipher that encrypts data in blocks of 64 bits. TripleDES executes the DES algorithm on a data block three times in succession by using a single, 56-bit key. This has the effect of encrypting the data by using a 168-bit key. TripleDES expands a single message to a maximum of 8 bytes. TripleDES is defined in the American National Standards Institute (ANSI) X9.52 specification.

Note: TripleDES is supported in SAS/SECURE and SSL. △

SASProprietary
: is a cipher that provides basic fixed encoding encryption services under all operating environments supported by SAS. Included in Base SAS, SASProprietary does not require additional SAS product licenses. The algorithm expands a single message to approximately one-third by using a 32-bit key.

Note: SASProprietary is supported only in SAS proprietary encryption services. △

IDEA (International Data Encryption Algorithm)
: is a 64-bit iterative block cipher that uses a 128-bit key. The speed of encryption for IDEA is similar to that of DES.

Note: IDEA is supported only in SSL. △

MD5 (Message Digest)
: is used for digital signature applications in which a large message must be securely compressed before being signed with a private key. The MD2, MD4, and MD5 family of algorithms share common structures, however, each design is unique. MD2 was optimized for 8-bit machines; MD4 and MD5 were designed for 32-bit machines. MD5 produces a 128-bit message digest from a message of arbitrary length.

Note: MD5 is supported only in SSL. △

Comparing Encryption Services

Note: SSL is supported only under the UNIX and Windows operating environments. △

Table 28.1 Summary of SSL, SAS/SECURE, and SASProprietary Features

Features	SSL	SAS/SECURE (all versions)	SASProprietary
License Required	No	Yes	No
Action Performed	Encryption and Authentication	Encryption only	Encryption only
Encryption Strength	Strong	Strong	Weak
Algorithms Supported	RC2, RC4, DES, TripleDES, IDEA, MD5, and others	RC2, RC4, DES, TripleDES	SASProprietary Fixed Encoding
Installation Required	Yes	Yes	No (part of Base SAS)
Operating Environments Supported	UNIX Windows	UNIX Windows z/OS	UNIX Windows z/OS OpenVMS Alpha
Enable with SAS Options	Yes	Yes	Yes
SAS Version Support	9 and later	8 and later	8 and later

Configuring and Using Encryption Services

Usually, a network administrator or a local system administrator installs and configures encryption services in the computing environment, as necessary. A user or applications programmer enables encryption in an application or in a SAS session by setting SAS system options. For more information, see “SAS/SECURE and SASProprietary Encryption Services” and “Secure Sockets Layer (SSL)”.

CHAPTER 29

SAS/SECURE and SASProprietary Encryption Services

Security System and Software Requirements

SAS/SECURE

SAS 9.1 supports SAS/SECURE under the following operating environments:

- z/OS
- UNIX
  - Compaq Tru64 UNIX
  - HP UX on Itanium 64-bit platform
  - HP UX on a 64-bit platform
  - Linux for Intel Architecture on a 32-bit platform
  - Solaris on a 64-bit platform
- Windows

SASProprietary

SAS 9.1 supports SASProprietary under the following operating environments:

- OpenVMS Alpha
- z/OS
- UNIX (all UNIX environments)
- Windows

Export Restrictions for SAS/SECURE

SAS/SECURE 9.1 is available to most commercial and government users inside and outside the U.S. However, some countries (for example, Russia, China, and France) have import restrictions on products that contain *encryption*, and the U.S. prohibits the export of encryption software to specific embargoed or restricted destinations. To comply with these regulations, SAS/SECURE is packaged according to the operating environment.

SAS/SECURE for z/OS and UNIX includes the following encryption algorithms:

- RC2 using 128-bit or 40-bit keys
- RC4 using 128-bit or 40-bit keys
- DES using 56-bit keys
- TripleDES using 168-bit keys

SAS/SECURE for Windows uses the encryption algorithms that are available in Microsoft CryptoAPI. The strength of the SAS/SECURE encryption algorithms under Windows depends on the strength of the encryption support in Microsoft CryptoAPI under Windows. For this reason, SAS/SECURE for Windows has very few export restrictions.

Examples: Setting Encryption in SAS/CONNECT

SAS/CONNECT Client

The following statements configure the client. The NETENCRYPTALGORITHM= option specifies the use of the RC4 algorithm.

```
options netencryptalgorithm=rc4;
options remote=unxnode comamid=tcp;
signon;
```

SAS/CONNECT Server

The following command starts a spawner on the machine that runs the server. The NETENCRYPT option specifies that encryption is required for all clients that connect to the spawner. The NETENCRYPTALGORITHM= option specifies the use of the RC4 algorithm for encrypting all network data. The SASCMD= option specifies the SAS startup command.

```
sastcpd -service spawner -netencrypt -netencryptalgorithm rc4 -sascmd mystartup
```

The spawner executes a UNIX shell script that executes the commands to start SAS.

```
#!/bin/ksh
#_________________
# mystartup
#_________________
. ~/.profile
sas dmr -noterminal -comamid tcp $*
```

CHAPTER

30 Secure Sockets Layer (SSL)

What Is SSL?

SSL (Secure Sockets Layer)

SSL is a protocol that provides secure network communications. Developed by Netscape Communications, SSL uses the encryption algorithms that were developed by RSA Security, Inc. and other cryptography experts.

In addition to providing encryption services, SSL performs client and server authentication and uses message authentication codes. SSL is supported by both Netscape Navigator and Internet Explorer. Many Web sites use this protocol to protect confidential user information, such as credit card numbers. URLs that require an SSL connection begin with https: instead of http:. The SSL protocol is application independent, which allows protocols such as HTTP, FTP, and Telnet to be transparently layered above it. SSL is optimized for HTTP.

Certification Authorities (CAs)

As e-business proliferates, there is a great need to ensure the confidentiality of business transactions over a network between an enterprise and its consumers, between enterprises, and within an enterprise. Cryptography products provide security services by exploiting digital certificates, public-key cryptography, private-key cryptography, and digital signatures. Certification authorities (CAs) create and maintain digital certificates, which also help preserve confidentiality.

Various commercial CAs, such as VeriSign and Thawte, provide competitive services for the e-commerce market. You can also develop your own CA by using products from companies such as RSA Security and Microsoft or from the Open Source Toolkit OpenSSL. From a trusted CA, members of an enterprise can obtain digital certificates to facilitate their e-business needs. The CA provides a variety of ongoing services to the business client that include handling digital certificate requests, issuing digital certificates, and revoking digital certificates.

Public and Private Keys

Public-key cryptography uses a public and a private key pair. The public key can be known by anyone, therefore, anyone can send a confidential message. The private key is confidential and known only to the owner of the key pair, therefore, only the owner can read the encrypted message. The public key is used primarily for encryption, but it can also be used to verify digital signatures. The private key is used primarily for decryption, but it can also be used to generate a digital signature.

Digital Signatures

A digital signature affixed to an electronic document or to a network data packet is like a personal signature that concludes a hand-written letter or that validates a credit card transaction. Digital signatures are a safeguard against fraud. A unique digital signature results from using a private key to encrypt a message digest. Receipt of a document that contains a digital signature enables the receiver to verify the source of the document. Electronic documents can be verified if you know where the document came from, who sent it, and when it was sent. Another form of verification comes from MACs, which ensure that a document has not been changed since it was signed.

Digital Certificates

Digital certificates are electronic documents that ensure the binding of a public key to an individual or an organization. Digital certificates provide protection from fraud.

Usually, a digital certificate contains a public key, a user's name, and an expiration date. It also contains the name of the certification authority (CA) that issued the digital certificate and a digital signature that is generated by the CA. The CA's validation of an individual or an organization allows that individual or organization to be accepted at sites that trust the CA.

Using SSL

Overview of SSL Set-Up Process

The details for installing and setting up SSL at your site are based on the operating environment and the digital certificate services software that you use. However, the following tasks are basic:

1. Access the appropriate software for installing and setting up digital certificate services under your operating environment.
2. Define a Certificate Authority (CA).
3. Request that digital certificates be generated by the CA for users, machines, and other CAs.
4. Store the digital certificates in a trusted repository.
5. View the properties of the generated digital certificates.
6. Start a server.
7. Connect to the server.

SSL for SAS

You can set SAS system options to use SSL in a SAS session. See the SAS options that are appropriate to your operating environment.

SSL for UNIX

System and Software Requirements for SSL under UNIX

The system and software requirements for using SSL under UNIX operating environments are:

- A computer that runs UNIX.
- Internet access and a Web browser such as Netscape Navigator or Internet Explorer.
- The TCP/IP communications access method.
- Access to the OpenSSL utility at www.openssl.org/source if you plan to use the OpenSSL CA.
- Knowledge of your site's security policy, practices, and technology. The properties of the digital certificates that you request are based on the security policies that have been adopted at your site.

Setting Up SSL under UNIX

Perform the following tasks to set up and use SSL:

1 Download and build SSL.

2 Create digital certificate requests.

3 Generate digital certificates from requests.

4 View the digital certificates.

5 Terminate the OpenSSL utility.

6 Create a trusted list of CAs.

Downloading and Building SSL under UNIX

If you want to use OpenSSL as your trusted Certificate Authority (CA), follow the instructions for downloading and building OpenSSL that are given at www.openssl.org/source. For complete documentation about the OpenSSL utility, visit www.openssl.org/docs/apps/openssl.html.

Information about alternative CAs and their Web sites follows:

□ For VeriSign, see www.verisign.com

□ For Thawte, see www.thawte.com

Creating Digital Certificate Requests under UNIX

To enable an SSL connection at your site, you must

□ obtain a digital certificate from a certification authority (CA).

□ create a digital certificate request from which a digital certificate is generated.

□ request one or more digital certificates for the CA (if you will be running your own CA), the server, and the client (optional).

The tasks that you perform to request a digital certificate for the CA, the server, and the client are similar, however, the values that you specify will be different.

In this example, Proton, Inc. is the organization that is applying for certification authority status by using OpenSSL. After Proton, Inc. becomes a CA, it can serve as a Certificate Authority for issuing digital certificates to clients (users) and servers on its network.

Perform the following tasks:

1 Select the **apps** subdirectory of the directory where OpenSSL was built.

2 Initialize OpenSSL.

```
$ openssl
```

3 Issue the appropriate command to request a digital certificate. (See Table 30.1 on page 287.) The functions of the arguments used in the commands are shown in Table 30.2 on page 287

Table 30.1 Open SSL Commands for Requesting a Digital Certificate

Request Certificate for	**OpenSSL Command**
CA	req -config ./openssl.cnf -new -out sas.req -keyout saskey.pem -nodes
Server	req -config ./openssl.cnf -new -out server.req -keyout serverkey.pem
Client	req -config ./openssl.cnf -new -out client.req -keyout clientkey.pem

Table 30.2 Arguments and Values Used in OpenSSL Commands

OpenSSL Arguments and Values	**Functions**
req	requests a certificate
-config ./openssl.cnf	specifies where the configuration details for the OpenSSL program are stored
-new	identifies the request as new
-out sas.req	specifies where the certificate request will be stored
-keyout saskey.pem	specifies where the private key will be stored
-nodes	prevents the private key from being encrypted

4 Informational messages are displayed and prompts for additional information appear according to the specific request.

To accept a default value, press the Return key. To change a default value, type the appropriate information and press the Return key.

Note: Unless the -NODES option is used in the OpenSSL command when creating a digital certificate request, OpenSSL will prompt you for a password before allowing access to the private key. △

The following is an example of a request for a digital certificate:

```
OpenSSL> req -config ./openssl.cnf -new -out sas.req -keyout saskey.pem -nodes
Using configuration from ./openssl.cnf
Generating a 1024 bit RSA private key
...........................++++++
.........................................++++++
writing new private key to 'saskey.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [US]:
State or Province Name (full name) [North Carolina]:
Locality Name (city) [Cary]:
Organization Name (company) [Proton INC.]:
Organizational Unit Name (department) [IDB]:
Common Name (YOUR name) []: Joe Bass
```

```
Email Address []:Joe.Bass@proton.com



Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
OpenSSL>
```

The request for a digital certificate is complete.

Note: For the server, the Common Name must be the name of the machine on which the server runs; for example, apex.serv.com. △

Generating Digital Certificates on UNIX

Perform the following tasks to generate digital certificates for a CA, a server, and a client.

1 Issue the appropriate command to generate a digital certificate from the digital certificate request. (See Table 30.3 on page 288.)

Table 30.3 OpenSSL Commands for Generating Digital Certificates under UNIX

Generate Certificate for	OpenSSL Command
CA	x509 req -in sas.req -signkey saskey.pem -out sas.pem
Server	ca -config ./openssl.cnf -in server.req -out server.pem -nodes
Client	ca -config ./openssl.cnf -in client.req -out client.pem

The functions performed by the OpenSSL arguments and values are shown in Table 30.4 on page 288.

Table 30.4 Arguments and Values Used in OpenSSL Commands on UNIX

OpenSSL Arguments and Values	Functions
x509	identifies the certificate display and signing utility
req	specifies that a certificate be generated from the request
ca	identifies the certificate authority utility
-config ./openssl.cnf	specifies where the configuration details for the OpenSSL utility are stored
-in filename.req	specifies where the input for the certificate request is stored
-out filename.pem	specifies where the certificate will be stored
-signkey saskey.pem	specifies the private key that will be used to sign the certificate that is generated by the certificate request

2 Informational messages are displayed and prompts for additional information appear according to the specific request.

To accept a default value, press the Return key. To change a default value, type the appropriate information, and press the Return key.

Sample dialog for creating a server digital certificate follows:

Note: The password is for the CA's private key. △

```
Using configuration from ./openssl.cnf
Enter PEM pass phrase: password
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName           :PRINTABLE:'US'
stateOrProvinceName   :PRINTABLE:'NC'
localityName          :PRINTABLE:'Cary'
organizationName      :PRINTABLE:'Proton, Inc.'
organizationalUnitName:PRINTABLE:'Development'
commonName            :PRINTABLE:'Server'
Certificate is to be certified until Oct 16 17:48:27 2003 GMT (365 days)
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries Data Base Updated
```

The subject's Distinguished Name is obtained from the digital certificate request.

A root CA digital certificate is self-signed. *Self-signed* means that the digital certificate is signed with the private key that corresponds to the public key that is in the digital certificate. Except for root CAs, digital certificates are usually signed with a private key that corresponds to a public key that belongs to someone else, usually the CA.

Viewing Digital Certificates

To view a digital certificate, issue the following command:

```
openssl> x509  -text -in filename.pem
```

A digital certificate contains data that was collected to generate the digital certificate timestamps, a digital signature, and other information. However, because the generated digital certificate is encoded (usually in PEM format), it is unreadable.

Terminating OpenSSL

To terminate OpenSSL, type **`quit`** at the prompt.

Creating a CA Trust List

After generating digital certificates for the CA, the server, and the client (optional), you must identify for the OpenSSL client application one or more CAs that are to be trusted. This list is called a *trust list*.

If there is only one CA to trust, specify the name of the file that contains the OpenSSL CA digital certificate, in the client application.

If multiple CAs are to be trusted, create a new file and copy-and-paste into it the contents of all the digital certificates for CAs to be trusted by the client application.

Use the following template to create a CA trust list:

```
Certificate for OpenSSL CA

-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----


Certificate for Keon CA

-----BEGIN CERTIFICATE-----

<PEM encoded certificate>

-----END CERTIFICATE-----


Certificate for Microsoft CA

-----BEGIN CERTIFICATE-----


-----END CERTIFICATE-----
```

Because the digital certificate is encoded, it is unreadable. Therefore, the content of the digital certificate in this example is represented as **`<PEM encoded certificate>`** . The content of each digital certificate is delimited with a **`-----BEGIN CERTIFICATE-----`** and **`-----END CERTIFICATE-----`** pair. All text outside the delimiters is ignored. Therefore, you might want to use undelimited lines for descriptive comments. In the preceding template, the file that is used contains the content of digital certificates for the CAs: OpenSSL, Keon, and Microsoft.

Note: If you are including a digital certificate that is stored in DER format, you must first convert it to PEM format. For more information, see "Converting between PEM and DER File Formats" on page 297. △

UNIX: SAS/CONNECT Example

UNIX: Starting a UNIX Spawner on a SAS/CONNECT Server

After digital certificates are generated for the CA, the server, and the client, and a CA trust list for the client is created, you can start a UNIX spawner program that runs on a server that SAS/CONNECT clients connect to.

For example:

```
% sastcpd -service unxspawn -netencryptalgorithm ssl
-sslcertloc /users/server/certificates/server.pem
-sslpvtkeyloc /users/server/certificates/serverkey.pem
-sslpvtkeypass starbuck1
-sslcalistloc /users/server/certificates/sas.pem
-sascmd /users/server/command.ksh
```

The following table explains the SAS commands that are used to start a spawner on a SAS/CONNECT single-user server.

Table 30.5 SAS Commands for Spawner Start-Up Tasks

SAS Command	Function
sastcpd	starts the spawner
-service unxspawn	specifies the spawner service (configured in the services file)
-netencryptalgorithm ssl	specifies the SSL encryption algorithm
-sslcertloc /users/server/certificates/server.pem	specifies the file path for the location of the server's certificate
-sslpvtkeyloc /users/server/certificates/ serverkey.pem	specifies the file path for the location of the server's private key
-sslpvtkeypass password	specifies the password to access the server's private key
-sslcalistloc /users/server/certificates/sas.pem	specifies the CA trust list
-sascmd /users/server/command.ksh	specifies the name of an executable file that starts a SAS session when you sign on without a script file.

Note: As an alternative to using the -SSLPVTKEYPASS option to protect the private key, you can use an unencrypted private key, and specify file system permissions that prevent read and write access to the file that contains the private key. To prevent the private key from being encrypted, use the –NODES option when requesting the digital certificate. △

An example of an executable file follows:

```
#!/bin/ksh
#-----------------------------------
# mystartup
#-----------------------------------

. ~/.profile
sas -dmr -noterminal $*
#-------------------------------
```

For complete information about starting a UNIX spawner, see the appropriate chapter in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Connecting a SAS/CONNECT Client to the UNIX Spawner

After a UNIX spawner is started on a SAS/CONNECT server, a SAS/CONNECT client can connect to it.

The following example shows how to make a client connection to a spawner that is running on a SAS/CONNECT server:

```
options netencryptalgorithm=ssl;
options sslcalistloc="/users/johndoe/certificates/cacerts.pem";
%machine=apex.server.com
%signon machine.spawner user=_prompt_;
```

The following table explains the SAS options that are used to connect to a SAS/CONNECT server.

Table 30.6 Client Access to a SAS/CONNECT Server

SAS Options	Client Access Tasks
NETENCRYPTALGORITHM=ssl	specifies the encryption algorithm
SSLCALISTLOC=cacerts.pem	specifies the CA trust list
SIGNON=server-ID.service	specifies the server and service to connect to
USER=_PROMPT_	prompts for the user ID and password to be used for authenticating the client to the server

The server-ID and the server's Common Name, which was specified in the server's digital certificate, must be identical.

For complete information about connecting to a UNIX spawner, see the chapter about UNIX:TCP/IP, the section about SAS/CONNECT client tasks, and the topic about signing on by using a spawner in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

SSL for Windows

System and Software Requirements for SSL under Windows

The system and software requirements for using SSL under the Windows operating environment are:

- □ A computer that runs Windows 2000 (or later).
- □ Depending on your configuration, it might be useful to have access to the Internet and a Web browser such as Netscape Navigator or Internet Explorer.
- □ The TCP/IP communications access method.
- □ Microsoft Certificate Services add-on software.
- □ The Microsoft Certificate Authority application (which is accessible from your Web browser) if you will run your own CA.
- □ In order for a SAS/CONNECT client session to connect to a SAS/CONNECT server session via a Windows spawner using SSL encryption, ensure that the client session runs on a machine that has a Trusted CA Certificate.

  The Windows spawner must run on a machine that has a Trusted CA Certificate and a Personal Certificate.
- □ Knowledge of your site's security policy, practices, and technology. The properties of the digital certificates that you request will depend on the security policies that have been adopted at your site.

Complete information about configuring your Windows operating environment for SSL is contained in the Windows installation documentation and at www.microsoft.com.

The following keywords might be helpful when searching the Microsoft Web site:

- □ digital certificate services
- □ digital certificate authority

□ digital certificate request
□ site security planning.

Digital Certificates Set-Up Process

The process for generating digital certificates under the Windows operating environment follows:

1 The user requests a digital certificate from a certificate authority (CA).
2 The CA issues a digital certificate.
3 The digital certificate is installed in a certificate store.

The tasks that you perform depend on the CA that you use:

See "Generating Digital Certificates Issued by Microsoft Certificate Authority" on page 293

See "Generating Digital Certificates Issued by a Certificate Authority That Is Not Microsoft" on page 294.

Generating Digital Certificates Issued by Microsoft Certificate Authority

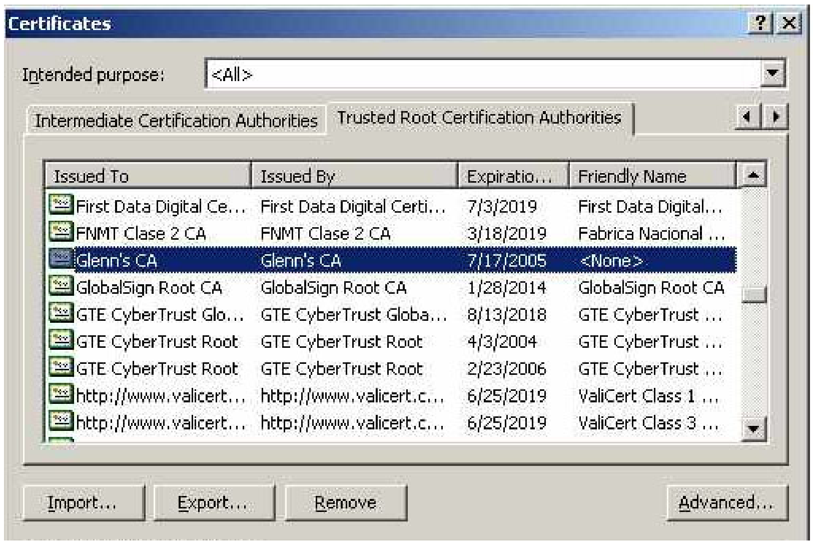
The following tasks are performed to generate digital certificates issued by the Microsoft Certificate Authority:

1 If you are running your own CA, the system administrator uses Microsoft Certificate Services to create an active Certificate Authority (CA).
2 The user

  a uses the Certificate Request wizard to request a digital certificate from an active enterprise CA. The Certificate Request wizard lists all digital certificate types that the user or computer is eligible to obtain.
  b selects a digital certificate type
  c specifies security options
  d submits the request to an active CA that is configured to issue the digital certificate.

  After the CA issues the requested digital certificate, the digital certificate is automatically installed in the Certificate Store. (See Display 30.1 on page 293 for an example.)

Display 30.1 Digital Certificate Installation in the Certificate Store


Certificates
Intended purpose: <All>
Intermediate Certification Authorities
Trusted Root Certification Authorities
Import...
Export...
Remove
Advanced...

Issued To	Issued By	Expiratio...	Friendly Name
First Data Digital Ce...	First Data Digital Certi...	7/3/2019	First Data Digital...
FNMT Clase 2 CA	FNMT Clase 2 CA	3/18/2019	Fabrica Nacional ...
Glenn's CA	Glenn's CA	7/17/2005	<None>
GlobalSign Root CA	GlobalSign Root CA	1/28/2014	GlobalSign Root CA
GTE CyberTrust Glo...	GTE CyberTrust Globa...	8/13/2018	GTE CyberTrust ...
GTE CyberTrust Root	GTE CyberTrust Root	4/3/2004	GTE CyberTrust ...
GTE CyberTrust Root	GTE CyberTrust Root	2/23/2006	GTE CyberTrust ...
http://www.valicert...	http://www.valicert.c...	6/25/2019	ValiCert Class 1 ...
http://www.valicert...	http://www.valicert.c...	6/25/2019	ValiCert Class 3 ...

Generating Digital Certificates Issued by a Certificate Authority That Is Not Microsoft

The following tasks are performed to generate digital certificates that are not issued by the Microsoft Certificate Authority:

1 the user requests a digital certificate from a CA and the digital certificate is issued.

2 the user imports digital certificates to a Certificate Store by using the Certificate Manager Import Wizard application from a Web browser. The digital certificates can be generated by using the Certificate Request wizard or any third-party application that generates digital certificates.

   Note: The Windows operating environment can import digital certificates that were generated in the UNIX operating environment. If you want to convert from PEM format (UNIX) to DER format (Windows) before importing, see “Converting between PEM and DER File Formats” on page 297. △

Importing Digital Certificates to a Certificate Store

Digital certificates that were issued by a third-party application can be imported to an appropriate Certificate Store, as follows:

Certificate Type	**Certificate Storage Location**
Client	Personal Certificate Store
Server	Personal Certificate Store
CA (self-signed)	Trusted Root Certification Authorities

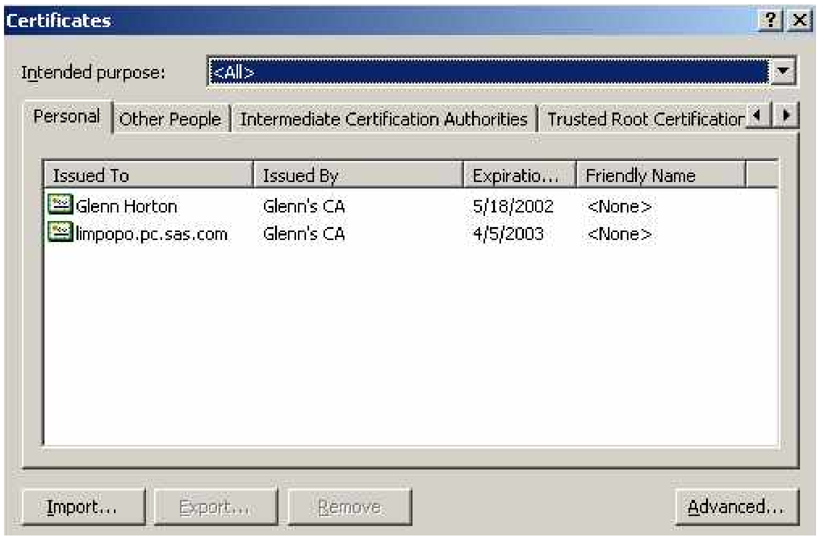
Perform the following tasks to import a digital certificate to a Certificate Store:

1 Access the Certificate Manager Import Wizard application from your Web browser. From the **`Tools`** pull-down menu, select

```
Tools -> Internet Options -> Content tab -> Certificates button
```

   Select the Personal tab in the Certificates window and specify which files you want to import to a Certificate Store. (See Display 30.2 on page 295)

Display 30.2 Digital Certificate Selections for a Personal Certificate Store


Certificates
Intended purpose: <All>
Personal | Other People | Intermediate Certification Authorities | Trusted Root Certificatior
Import... | Export... | Remove | Advanced...

Issued To	Issued By	Expiratio...	Friendly Name
Glenn Horton	Glenn's CA	5/18/2002	<None>
limpopo.pc.sas.com	Glenn's CA	4/5/2003	<None>

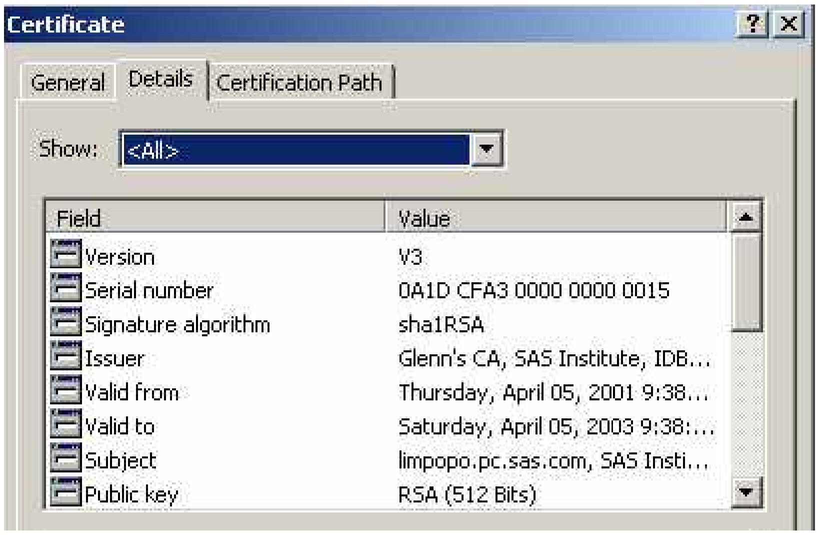
2 Click Import and follow the instructions to import digital certificates.

Repeat this task in order to import the necessary digital certificates for the CA, the server, and the client, as appropriate.

3 After you have completed the selections for your personal Certificate Store, select the appropriate tab to view your selections.

4 To view the details about a digital certificate, select the digital certificate and click View. The results are shown in Display 30.3 on page 295.

Display 30.3 Digital Certificate Details Tab



Field	Value
Version	V3
Serial number	0A1D CFA3 0000 0000 0015
Signature algorithm	sha1RSA
Issuer	Glenn's CA, SAS Institute, IDB...
Valid from	Thursday, April 05, 2001 9:38...
Valid to	Saturday, April 05, 2003 9:38:...
Subject	limpopo.pc.sas.com, SAS Insti...
Public key	RSA (512 Bits)

Windows: SAS/CONNECT Example

Starting a Windows Spawner on a Single-User SAS/CONNECT Server

After digital certificates for the CA, the server, and the client have been generated and imported into the appropriate Certificate Store, you can start a spawner program that runs on a server that SAS/CONNECT clients connect to.

An example of how to start a Windows spawner on a SAS/CONNECT server follows:

```
spawner -security -netencryptalgorithm ssl -sslcertsubj "apex.pc.com" -sascmd mysas.bat
```

Table 30.7 on page 296 shows the SAS commands that are used to start a spawner on a SAS/CONNECT single-user server.

Table 30.7 SAS Commands for Spawner Start Up

SAS Command	Function
spawner	starts the spawner.
-security	specifies the requirement that a client provide a username and password to access the spawner
-netencryptalgorithm ssl	specifies the SSL encryption algorithm
-sslcertsubj "apex.pc.com"	specifies the subject name that is used to search for a certificate from the Microsoft Certificate Store
-sascmd mysas.bat	specifies the name of an executable file that starts a SAS session when you sign on without a script file

In order for the Windows spawner to locate the appropriate server digital certificate in the Microsoft Certificate Store, you must specify the -SSLCERTSUBJ system option in the script that is specified by the -SASCMD option. -SSLCERTSUBJ specifies the subject name of the digital certificate that SSL should use. For more information, see "SSLCERTSUBJ= System Option" on page 39.

If the Windows spawner is started as a service, the -SERVPASS and -SERVUSER options must also be specified in the Windows spawner start-up command in order for SSL to locate the appropriate CA digital certificate.

For complete information about starting a Windows spawner, see "Windows Spawner" in *Communications Access Methods for SAS/CONNECT and SAS/SHARE*.

Connecting to a Windows Spawner on a SAS/CONNECT Server

After a spawner has been started on a SAS/CONNECT server, a SAS/CONNECT client can connect to it.

An example of how to make a client connection to a Windows spawner that is running on a SAS/CONNECT server follows:

```
options comamid=tcp netencryptalgorithm=ssl;
%let machine=apex.pc.com
signon machine user=_prompt_;
```

Table 30.8 on page 297 shows the SAS options that are used to connect to a Windows spawner that runs on a SAS/CONNECT server.

Table 30.8 Client Access to a SAS/CONNECT Server

SAS Options	Function
COMAMID=tcp	specifies the TCP/IP access method
NETENCRYPTALGORITHM=ssl	specifies the encryption algorithm
SIGNON=server-ID	specifies which server to connect to
USER=_PROMPT_	prompts for the user ID and password to be used for authenticating the client to the server

The server-ID and the server's Common Name, which was specified in the server's digital certificate, must be identical.

Converting between PEM and DER File Formats

By default, OpenSSL files are created in PEM (Privacy Enhanced Mail) format. SSL files that are created in Windows operating environments are created in DER (Distinguished Encoding Rules) format.

Under Windows, you can import a file that is created in either PEM or DER format. However, a digital certificate that is created in DER format must be converted to PEM format before it can be included in a trust list under UNIX.

An example of converting a server digital certificate from PEM input format to DER output format follows:

```
OpenSSL> x509 -inform PEM -outform DER -in server.pem -out server.der
```

An example of converting a server digital certificate from DER input format to PEM output format follows:

```
OpenSSL> x509 -inform DER -outform PEM -in server.der -out server.pem
```

PART 8

Appendices

APPENDIX 1 Cross-Architecture Issues

Translation of SAS Data between Machines That Represent Data Differently

Overview of Data Translation between Machines

SAS/CONNECT clients and servers can access SAS data and programs from each other, despite differences in how data is represented on machines 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, bypassing 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 machine to the representation of the client machine.

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 machine to the representation of the client machine.
- □ When the data is updated (and stored) on the server, it is translated from the representation of the client machine back to the representation of the server machine.

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 where the server data comes from and how the data will be processed. The following example shows how data is read (and translated) from the server and processed, with results being 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 download large quantities of server data. RLS is used here to illustrate the possibility for the loss of precision across machines that represent numeric data differently. △

For details about how to access a remote file system, see Chapter 18, “Remote Library Services (RLS),” on page 189.

Data Transfer Services

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 23, “Using Data Transfer Services,” on page 215.

The translation process for transferring data varies according to the SAS release.

Translation of Version 8 and Later Releases

In Version 8 and later releases, translation occurs only once for each data transfer between a client and a server that run on machines whose architectures are different from the 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 'serv-library';
   proc download data=serv-libref.data-set
      out=client-libref.data-set;
endrsubmit;
```

Version 6 Translation

In Version 6, translation occurs twice for each data transfer between a client and a server that run on machines whose architectures are different from the other.

1 The data is translated from the source machine's native format to transport format.
2 The data that is represented in transport format is translated to the target machine'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 machine format when copied to a specified client location.

```
libname client-libref 'client-library';
rsubmit;
   libname serv-libref 'serv-library';
   proc download data=serv-libref.data-set
      out=client-libref.data-set;
endrsubmit;
```

Translation of Floating-Point Numbers between Machines

Loss of Numeric Precision and Magnitude

If you move SAS data between a client and a server session that run on machines 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. In general,

- the *larger* the number, the *less* concern there is for loss of precision.
- the *smaller* the number, the *more* concern there is for loss of precision.

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, but if you have data with extremely large values or extremely small fractions 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 A1.1 Approximate Value Ranges by Operating Environment

Operating Environment	**Minimum Value**	**Maximum Value**
OpenVMS Alpha	2.3E-308	1.8E+308
OpenVMS VAX	2.9E-39	1.7E+38
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.

Translation of Character Data in International Environments

If SAS/CONNECT is used in an international environment, the client and server sessions might use different encodings that reflect their national languages and customs. For example, a server session and server data might be encoded for Latin (language) and Windows (encoding method). A client session might be encoded for German EBCDIC. Data that flows across architectures and encodings that are sent from a server to a client and back to the server is translated twice.

For complete details about internationalizing a SAS/CONNECT client/server environment, see the *SAS National Language Support (NLS): User's Guide*.

APPENDIX

2

SAS/CONNECT Cross-Version Issues

Factors Affecting File Access

SAS files (data and applications) that were created by using SAS releases later than Version 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 Version 6) are dramatically different from older SAS releases (Versions 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 file
    - Catalog
    - View
  - SAS/CONNECT service
    - Remote Library Services (RLS)

      CAUTION:
      RLS in SAS/CONNECT 9 and later is not backward compatible with Version 6 SAS files. SAS/CONNECT 9 clients cannot access Version 6 SAS/CONNECT servers. Version 6 SAS/CONNECT clients cannot access SAS/CONNECT 9 servers. △

    - Compute Services
    - File Transfer Services.

Newer SAS Release Access to Version 6 Data

Here is what you can expect as you access data and move applications between newer releases and Version 6.

Table A2.1 Accessing Version 6 Data

SAS	Access to Version 6 Data Sets	Access to Version 6 Views and Catalogs
Version 9	No Read, Write, or Update	No Read, Write, or Update
Version 8	Read, Write, and Update (New SAS features cannot be written to Version 6 data sets).	Read only

For information about SAS releases that relates to single-user SAS mode, see *SAS Language Reference: Concepts*. For information that relates to SAS/SHARE software, see the *SAS/SHARE User's Guide*.

Features Exclusive to SAS Releases after Version 6

File Format Features

The file format features of newer SAS releases and Version 6 are incompatible. File format features of the newer releases follow:

- long data set labels
- long variable labels
- long variable names.

However, in order to maintain filename compatibility between the newer and older SAS releases, SAS/CONNECT will apply truncation rules to file format names. Truncation enables you to take advantage of the features of the newer SAS releases while continuing to access Version 6 SAS files in a mixed-version environment.

New Features Incompatible with Version 6

These new features in SAS cannot be modified to make data files compatible with Version 6:

- □ generation data sets
- □ integrity constraints.

Any attempt to access data files that contain these features will fail. For complete details about new features, see *SAS Language Reference: Concepts*.

File Transfer Services: Truncating Long Names and Labels

The newer SAS releases support longer names and labels than the maximum length supported in Version 6. The longer names and labels are stored in Versions 8 (or later) data files, which make those data files incompatible with Version 6 data files. SAS/CONNECT implements a set of truncation rules to convert data files that contain long names and labels into Version 6 data files.

The UPLOAD or DOWNLOAD procedures apply the truncation rules when transferring a SAS file

- □ from a Version 8 (or later) SAS session to a Version 6 SAS session.
- □ between two sessions (each running Version 8 or later) to produce a Version 6 data file.

  Note: To produce a Version 6 data file explicitly, specify VALIDVARNAME=V6 in the SAS session that the data file 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 Version 6.

Table A2.2 Version 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. Additional truncation rules follow:

Table A2.3 Truncation Rules To Resolve Conflicting Variable Names

Truncation Rule	Example
The first name that has more than 8 characters is truncated to 8 characters.	STOCKNUMBER53 is truncated to STOCKNUM.
The next name that has more than 8 characters is truncated to 8 characters. If it conflicts with an existing variable name, it is truncated to 7 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: Accessing Data Files in a Mixed Cross-Version Library

Separating Older SAS Files from Newer SAS Files

Whenever possible, keep older SAS files (Version 6) and newer SAS files (created using SAS releases after Version 6) in separate physical locations. Segregation of release-specific files avoids confusion about what files can be accessed when using RLS.

Specifying an Engine to Locate Release-Specific Files in a Mixed Library

Your ability to access a specific 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 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 Version 6 files and Version 8 files, a SAS session that runs a newer release of SAS can use the V6 engine to access only the Version 6 SAS files in that library.

CAUTION:
A SAS 9 session cannot access Version 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 Version 8 files can be accessed. The Version 6 files are not recognized in the SAS 9 or Version 8 session.
However, if the V6 engine is specified, the Version 6 files can be accessed. The SAS 9 or Version 8 files are not recognized.
In the following example, the libref V8LIB accesses only SAS 9 or Version 8 files.

```
libname v8lib v8 'SAS-data-library';
```

In the following example, the libref V9LIB accesses only SAS 9 or Version 8 files.

```
libname v9lib v9 'SAS-data-library';
```

In the following example, the libref V6LIB accesses only Version 6 files.

```
libname v6lib v6 'SAS-data-library';
```

Determining the Version of SAS Used to Create a File

To find out which version of SAS was used to create a file, examine the filename extension.

Filename extensions for files that are created in the Windows operating environment follow:

Table A2.4 Filename Extensions for a Windows Machine

File Type	Version 6 Filename Extension	SAS 9 or Version 8 Filename Extension
Data Set	sd2	sas7bdat
Catalog	sc2	sas7bcat
View	sv2	sas7bvew

Concatenating Libraries

In order to expand the scope of file access from a single library to multiple libraries, use library concatenation. With an expanded scope, you can perform operations on either Version 6 files or Version 8 files that span multiple libraries.

CAUTION:
A SAS 9 session cannot concatenate libraries that contain Version 6 files. △

Here is an example of library concatenation:

```
libname v6lib v6 'SAS-data-library';
libname v8lib v8 'SAS-data-library';
libname catlib (v8lib v6lib);
```

Note: *SAS-data-library* must be the physical name that is recognized by the operating environment. △

The first LIBNAME statement assigns the libref V6LIB to a SAS data library that is accessed with the V6 engine. The V6 engine recognizes only files that are appended with a Version 6 filename extension.

The second LIBNAME statement assigns the libref V8LIB to a SAS data library that is accessed with the V8 engine. The V8 engine recognizes only files that are appended with a Version 8 filename extension.

The third LIBNAME statement assigns the libref CATLIB to concatenated libraries that are referenced by the librefs V8LIB 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 file exists in both libraries and you delete that file, the file in the first library (for example, STOCK.SAS7BDAT in V8LIB) is deleted. If V6LIB precedes V8LIB in the library concatenation statement (for example, STOCK.SD2 in

V6LIB), that file is deleted. If the specified file exists in only one library, that file is deleted.

Accessing SAS Data Sets

Limitations

Accessing data that is stored in a SAS data set is a fundamental operation in SAS. It is important to 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 Version 6.

Version 6 Client Accessing a Version 8 (or later) Server

This table summarizes the limitations of a Version 6 client that accesses data sets on a Version 8 (or later) server in a cross-version environment.

Table A2.5 Limitations for Accessing Data Sets on Version 8 (or Later) from Version 6

SAS/CONNECT Service	**Version 6 Client Connecting to SAS 9 Server**	**V6 Client Connecting to Version 8 Server**
Remote Library Services	A Version 6 client cannot read, write, or update SAS 9 data files on a SAS 9 server.	If Version 8 files on a Version 8 server do not implement new features, a Version 6 client can read, write, or update Version 8 data files on a Version 8 server.
Data Transfer Services	All data formats are automatically converted when uploading or downloading a Version 6 file to a SAS 9 or Version 8 target. If SAS 9 or Version 8 data files do not contain new features, they can be downloaded to a Version 6 target. Truncation rules are applied.	
Compute Services	A Version 6 client can remote submit a SAS program to a SAS 9 or Version 8 server. The data files that are referenced in the remote submit blocks can be SAS 9, Version 8, or Version 6 data files.	

Version 8 (or Later) Client Accessing a Version 6 Server

This table summarizes the limitations of a Version 8 (or later) client that accesses data sets on a Version 6 server in a cross-version environment.

Table A2.6 Limitations for Accessing Data Sets on Version 6 from Version 8 (or Later)

SAS/CONNECT Service	SAS 9 Client Connecting to a Version 6 Server	Version 8 Client Connecting to a Version 6 Server
Remote Library Services	A SAS 9 client cannot read, write, or update Version 6 data files on a Version 6 server.	If Version 6 data files do not implement new features, a Version 8 client can read, write, or update Version 6 data files on a Version 6 server.
Data Transfer Services	All data formats are automatically converted when uploading or downloading a Version 6 file to a SAS 9 or Version 8 target. If SAS 9 or Version 8 data files do not contain new features, they can be downloaded to a Version 6 target. Truncation rules are applied.	
Compute Services	A SAS 9 or Version 8 client can remote submit a SAS program to a Version 6 server. The data files that are referenced in the remote submit blocks can be formatted as SAS 9, Version 8, or Version 6 files.	

Accessing SAS Views

Limitations

There are limitations and restrictions when accessing data views in a cross-version environment. Types of data views follow:

- DATA step
- PROC SQL
- SAS/ACCESS

Note: SAS/CONNECT uses the data that the view references, but not the view itself. △

Version 6 Client Accessing a Version 8 (or Later) Server

This table summarizes the limitations of a Version 6 client that accesses data views on a Version 8 (or later) server in a cross-version environment.

Table A2.7 Limitations for Accessing Data Views on Version 8 (or Later) from Version 6

SAS/CONNECT Service	Version 6 Client Connecting to SAS 9 Server	Version 6 Client Connecting to Version 8 Server
Remote Library Services	A Version 6 client cannot read, write, or update a SAS 9 DATA step view, a SAS 9 PROC SQL view, or a SAS 9 SAS/ACCESS view on a SAS 9 server.	For Version 8 DATA step views and Version 8 PROC SQL views, if the view is processed at the Version 8 server (RMTVIEW=YES in the LIBNAME statement), the Version 6 client has only read access. For Version 8 SAS/ACCESS views, the Version 6 client has read, write, and update access.
Data Transfer Services	For PROC SQL views, a Version 6 client can upload and download a PROC SQL view between a SAS 9 or Version 8 server by using the DATA= option to specify the data file (or the INLIB= option to specify the library) associated with the view to transfer.	
Compute Services	For SAS data views, a Version 6 client can remote submit a SAS program that references SAS data views to a SAS 9 or Version 8 server. The data views that are referenced in remote submit blocks can be SAS 9, Version 8, or Version 6 data files.	

Version 8 (or Later) Client Accessing a Version 6 Server

This table summarizes the limitations of a Version 8 (or later) client that accesses data views on a Version 6 server in a cross-version environment.

Table A2.8 Limitations for Accessing Data Views on Version 6 from Version 8 (or Later)

SAS/CONNECT Service	SAS 9 Client Connecting to a Version 6 Server	Version 8 Client Connecting to a Version 6 Server
Remote Library Services	A SAS 9 client cannot read, write, or update a Version 6 DATA step view, a Version 6 PROC SQL view, or a Version 6 SAS/ACCESS view on a Version 6 server.	For Version 6 DATA step views and Version 6 PROC SQL views, if the view is processed at the server (RMTVIEW=YES in the LIBNAME statement), the Version 8 client has read access only. For Version 6 SAS/ACCESS views, the Version 8 client has read, write, and update access.
Data Transfer Services	A SAS 9 or Version 8 client can upload data that is associated with a view to a Version 6 server. Names of files that are transferred to a Version 6 server are truncated, following truncation rules.	
Compute Services	A SAS 9 or Version 8 client can remote submit a SAS program that references Version 6 SAS data views to a Version 6 server. The data files that are referenced in remote submit blocks cannot implement new features.	

Accessing Catalogs

Limitations

There are limitations and restrictions when accessing catalogs in a cross-version environment.

CAUTION:

A SAS 9 or Version 8 SAS session cannot read Version 6 catalogs on AIX/RS6000. Use the CPORT and CIMPORT procedures to migrate Version 6 catalogs into a SAS 9 or Version 8 environment on AIX. △

Version 8 (or later) catalog entry types (alphabetized horizontally) that are compatible with Version 6 include:

AFCBT	AFGO	DEVMAP
FONT	FONTLIST	KEYMAP
KEYS	LOG	OUTPUT
SOURCE	TEMPLATE	TRANTAB

Version 6 Client Accessing a Version 8 (or Later) Server

This table summarizes the limitations of a Version 6 client that accesses catalogs on a Version 8 (or later) server in a cross-version environment.

Table A2.9 Limitations for Accessing Catalogs on Version 8 (or Later) from Version 6

SAS/CONNECT Service	**SAS 9 Client Connecting to Version 6 Server**	**Version 8 Client Connecting to SAS 9 Server**
Remote Library Services	A Version 6 client cannot read, write, or update a SAS 9 catalog on a SAS 9 server.	A Version 6 client can read a Version 6 catalog on a Version 8 server. A Version 6 client can read, write, and update a Version 8 catalog that does not contain new features.
Data Transfer Services	A Version 6 client can upload a Version 6 catalog from a SAS 9 or Version 8 server. The uploaded catalog is converted to SAS 9 or Version 8 format. A Version 6 client can download a SAS 9 or Version 8 catalog if the entry type does not contain new features.	
Compute Services	A Version 6 client can remote submit a SAS program that references a SAS catalog to a SAS 9 or Version 8 server.	

Version 8 (or Later) Client Accessing a Version 6 Server

This table summarizes the limitations of a Version 8 (or later) client that accesses catalogs on a Version 6 server in a cross-version environment.

Table A2.10 Limitations for Accessing Catalogs on Version 6 from Version 8 (or Later)

SAS/CONNECT Service	SAS 9 Client Connecting to a Version 6 Server	Version 8 Client Connecting to a Version 6 Server
Remote Library Services	A SAS 9 client cannot read, write, or update a Version 6 catalog on a Version 6 server.	A Version 8 client can read from and write to a Version 6 catalog on a Version 6 server. A Version 8 client can write a Version 6 catalog from one Version 6 library to another Version 6 library by using PROC COPY.
Data Transfer Services	A SAS 9 or Version 8 client can download a Version 6 catalog from a Version 6 server. A SAS 9 or Version 8 server can upload a Version 6 catalog from a SAS 9 or Version 8 server if the entry type does not contain new features. A SAS 9 or Version 8 client cannot create a Version 6 catalog entry by using PROC UPLOAD.	
Compute Services	A SAS 9 or Version 8 client can remote submit a SAS program that references a SAS catalog to a Version 6 server. The entries that are referenced in the remote submit blocks cannot contain new features.	

File Format Translation Algorithms

Version 6 Translation

In Version 6, translation occurs twice for each data transfer between a client and a server that run on machines whose architectures are incompatible.

1 The data is translated from the source machine's native format to transport format.

2 The data that is represented in transport format is translated to the target machine's native format.

Version 8 (and Later) Translation

In Version 8 and later releases of SAS, translation occurs only once for each data transfer between a client and a server that run on machines 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.

APPENDIX 3 Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *SAS/SHARE User's Guide*
- *Communications Access Methods for SAS/CONNECT and SAS/SHARE*
- *SAS Language Reference: Dictionary*
- SAS Companion that is specific to your operating environment

For a complete list of SAS publications, see the current *SAS Publishing Catalog*. To order the most current publications or to receive a free copy of the catalog, contact a SAS representative at

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513
Telephone: (800) 727-3228*
Fax: (919) 677-8166
E-mail: **sasbook@sas.com**
Web address: **support.sas.com/pubs**
* For other SAS Institute business, call (919) 677-8000.

Customers outside the United States should contact their local SAS office.

Glossary

access method
See communications access method.

ASCII (American Standard Code for Information Interchange)
a 7-bit character coding scheme (8 bits when a parity check bit is included) that includes graphic (printable) and control (nonprintable) codes.

asynchronous processing
a type of server processing that enables you to submit one or more tasks to one or more server sessions to execute in parallel and to resume client processing immediately. You do not wait for the server processing to complete before control is returned to the client session. Always maintaining control of the client session, you can submit multiple tasks to multiple server sessions that execute in parallel, thus making efficient use of time and resources.

autoexec file
a file that contains SAS statements that are executed automatically when you invoke SAS. You can use the autoexec file to specify some SAS system options, as well as to assign librefs and filerefs.

authentication
the process of verifying the identity of an entity. This is typically done so that a system can use the identity to make decisions about the entity's access to protected resources.

batch mode
a method of executing SAS programs in which a file that contains SAS statements plus any necessary operating environment commands is submitted to the computer's batch queue. After you submit the program, control returns to your terminal or workstation, where you can perform other tasks. Batch mode is sometimes referred to as running in the background. The program output can be written to files or printed on an output device.

binary
the base 2 number system. A binary digit can have one of two values: 0 or 1. A binary digit is called a bit and is considered to be off when its value is 0, and on when its value is 1.

block
the grouping of statements between a logical beginning and ending statement. For example, the statements between an RSUBMIT and an ENDRSUBMIT statement.

break signal
a signal that interrupts an executing program. If you send a break signal while an RSUBMIT, SIGNON, or SIGNOFF command executes, the link software will display a Break window to enable you to decide how to proceed by using special Break window commands.

Break window
a special class of windows for SAS/CONNECT. Break windows enable you to handle error conditions and interruptions that you cause by issuing a break signal.

buffer
a portion of the computer's memory that is used for special holding purposes or processes. For example, a buffer might store information before sending it to main memory for processing. Or a buffer might hold data after it is read or before it is written.

catalog
See SAS catalog.

catalog entry
See entry type and SAS catalog entry.

CEDA (Cross-Environment Data Access)
a feature of SAS software that enables a SAS data file that was created in any directory-based operating environment (for example, Solaris, Windows, HP-UX, OpenVMS) to be read by a SAS session that is running in another directory-based environment. You can access the SAS data files without performing any intermediate conversion steps.

character set
a standardized way of representing alphabetic, numeric, and control characters. The most widely used character sets are ASCII and EBCDIC.

character set translation table
a file that contains ASCII-to-EBCDIC or EBCDIC-to-ASCII translation information.

client
the side of the client/server relationship that requests the delivery of a particular type of information. The client side is often considered the workstation or personal computer from which the request is made. Examples of information that is requested might be computing resources and files (SAS files and external files). Implicit in the client/server relationship is the network. See also server, SAS/CONNECT client, and SAS/CONNECT server.

client session
See SAS/CONNECT client

command file
a file that contains operating environment commands to be executed in sequence.

Communication Services Break Handler window
one of two possible windows that are displayed when a server session is interrupted by a break signal or when there is an error in a statement that is submitted to the server. This window offers the following selections: *continue, disconnect link, debug, re-send, snapshot screen, type, subset command*, and *invoke the Application Break Handler*. See also SAS/CONNECT attention handler window.

communications access method
the method that a client uses to communicate with a server. You can use the COMAMID= system option to specify the communications access method.

Compute Services (CS)
SAS services for distributed applications. These services use server computing resources (hardware, software, and data) to execute an application more efficiently than execution on a single system would be. See also distributed application.

configuration file
an external file that contains the SAS system options that are put into effect when you invoke SAS.

control character
a character that is used for control purposes rather than for information exchange. Control characters are usually nonprintable.

checksum
one or more characters appended to the end of a data block for error checking purposes.

cryptography
an area of research that uses mathematics to secure information.

Data Transfer Services (DTS)
a feature of SAS/CONNECT that transfers files between a client and a server. DTS converts the data as needed if the client and the server do not use identical machine architectures and SAS releases. See also download, upload.

data translation
the conversion of data from one machine architecture to another machine architecture.

distributed application
an application, typically implemented in a client/server environment, whose components are distributed for execution over various nodes of the network. Distributing applications can significantly enhance overall performance. See also client/server.

download
to copy a file from the server to the client. See also Data Transfer Services (DTS).

EBCDIC (Extended Binary Coded Decimal Interchange Code)
an 8-bit character coding scheme that includes graphic (printable) and control (nonprintable) codes.

encryption
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.

engine
a component of SAS that reads from or writes to a file. Each engine allows SAS to access files in a particular format. There are several types of engines. The REMOTE engine is an example of an engine.

entry
See SAS catalog entry.

entry type
a characteristic of a SAS catalog entry that identifies its structure and attributes to SAS. When you create an entry, SAS automatically assigns the entry type as part of the name.

external database
a database that stores data that is not part of the SAS System, for example DB2, Oracle, and SYBASE.

external file
(1) a file that is maintained by the operating environment that SAS can read data from and route output to. External files can contain raw data, SAS programming statements, procedure output, or output created by the PUT statement. (2) in a DATA step, a file that the SAS System can read by using INFILE and INPUT statements or a file that the SAS System can write to by using FILE and PUT statements.

file specification
the name of an external file. This name is the name that the operating environment uses to recognize the file. On directory-based systems, the file specification can be either the complete pathname or the relative pathname from the current working directory.

fileref
a name temporarily assigned to an external file that identifies it to SAS.
Do not confuse filerefs with librefs. Filerefs are used for external files; librefs are used for SAS data libraries. See also libref.

GRLINK
a special device driver that enables you to execute graphics statements on the server and display graphs on the client. GRLINK must be installed on the server to enable this functionality.

interactive line mode
a method of running SAS programs without using the SAS windowing environment. SAS processes each line immediately after you submit it.

libref
a name that is temporarily associated with a SAS data library.

libref inheritance
a feature of SAS/CONNECT that enables server sessions to use client-defined librefs. See also libref.

line mode
See interactive line mode.

link
intercomputer communication in which one computer is a client and the other is a server. The link can be used to transfer files or to distribute processing to computers other than the client.

Local Area Network (LAN)
a group of computers and other devices that are dispersed over a geographically limited area and connected by a communications link that enables any device to interact with any other that is on the network.

macro facility
a part of SAS that is used for extending and customizing SAS software. A macro facility is also used to reduce the amount of text that must be entered to do common tasks. It consists of the macro processor and the macro language.

macro variable
a variable belonging to the macro language whose value is a string that remains constant until you change it. A macro variable is also called a symbolic variable.

member
(1) a file in a SAS data library. (2) under z/OS, a single component of a partitioned data set. (3) under OpenVMS, a component of an OpenVMS text library.

member name
(1) a name that is given to a SAS file in a SAS data library. Member names can reference such file types as a SAS data set, a catalog, an access descriptor, or a stored program. (3) under z/OS, the name of a single component of a partitioned data set. (5) under OpenVMS, member name is synonymous with filename for files that are stored in a SAS data library.

member type
a name that is assigned by SAS that identifies the type of information that is stored in a SAS file. Member types include DATA, CATALOG, MDDB, and VIEW.

MP(Multi-Processing) CONNECT
a feature of SAS/CONNECT that uses multiple CPUs together to process tasks. Multi-processing can be used within an operating environment with SMP hardware, across operating environments, or both. See also SMP hardware.

noninteractive mode
a method of running SAS programs in which you prepare a file of SAS statements and submit the program to the operating environment. The program runs immediately and occupies your current session.

non-U.S. keyboard
a keyboard that is not a standard U.S. keyboard. Non-English language keyboards often have characters that are not found on U.S. keyboards and might not have some characters that are found on U.S. keyboards.

operating environment
a computer, or a logical partition of a computer, and the resources (such as an operating system and other software and hardware) that are available to the computer or partition.

outbound packet
a unit of bytes that is transmitted from a server to a client. See also packet.

packet
a grouping of printable characters, a sequence number, and a checksum that are transmitted over the link as a unit. The SAS/CONNECT clients and servers use these specially formatted packets to communicate with each other.

permanent SAS data library
a library that is not deleted when the SAS session terminates and is available for subsequent SAS sessions. Unless the USER libref is defined, you use a two-level name to access a file in a permanent library. The first-level name is the libref, and the second-level name is the member name.

permanent SAS file
a SAS file in a library that is not deleted when the SAS session or job terminates.

piping
an extension of the MP CONNECT functionality that enables you to run multiple dependent processes asynchronously. Piping improves performance for some tasks by writing outputs to TCP/IP ports instead of to disk.

REMOTE engine
a SAS library engine that enables a user's SAS session to access data by communicating with another SAS session. See also Remote Library Services (RLS), server session.

Remote Library Services (RLS)
SAS services for distributed applications. These services provide transparent access to remote data libraries and move data through the network as the client session requests it. The data must pass through the network if it is needed subsequently by client processing. See also distributed application.

remote submit
to use the RSUBMIT command or statement to submit statements that are entered in a client session for execution by a server session.

return code
a code that is passed to the operating environment that reports whether a command or job step has executed successfully.

SAS catalog
a SAS file that stores many different kinds of information in units that are called catalog entries. A single SAS catalog can contain several different types of catalog entries. Some catalog entries contain system information such as key definitions. Other catalog entries contain application information such as window definitions, help windows, formats, informats, macros, or graphics output.

SAS catalog entry
a separate storage unit within a SAS catalog. Each entry has an entry type that identifies its structure to SAS. See also entry type.

SAS command
a command that invokes SAS. This command may vary based on operating environment and site. See also SAS invocation.

SAS Component Language (SCL)
a programming language provided in SAS/AF and SAS/FSP software to: develop interactive applications that manipulate SAS data sets and external files; display tables, menus, and selection lists; and generate SAS source code and submit it to SAS for execution.

SAS/CONNECT attention handler window
one of two possible windows that are displayed when a server session is interrupted by a break signal. This window offers the following selections: abort current server processing or continue processing the current remote submit. See also Communication Services Break Handler window.

SAS/CONNECT client
a SAS/CONNECT session that acts as a client. The user that runs a SAS/CONNECT client requests services from a SAS/CONNECT server that can run on a remote single-processor machine or on a local or remote SMP machine. Services supported are Remote Library Services, which enables access to SAS files; Compute Services, which exploits fast processing resources; and Data Transfer Services, which allows the upload or download of selected data for processing. See also client, server, SAS/CONNECT server, Remote Library Services, Compute Services, and Data Transfer Services.

SAS/CONNECT server
a SAS/CONNECT session that acts as a server. The SAS/CONNECT server runs a SAS session on a machine that receives requests for services from a SAS/CONNECT client. A server can run on a remote single-processor machine or on a local or remote

SMP machine. The services that are supported are Remote Library Services, which enables access to SAS files; Compute Services, which exploits fast processing resources; and Data Transfer Services, which allows the upload or download of selected data for processing. See also client, server, SAS/CONNECT client, Remote Library Services, Compute Services, and Data Transfer Services.

SAS data file
a SAS data set that contains both the data values and the descriptor information.

SAS data library
a collection of one or more SAS files that is recognized by SAS and is referenced and stored as a unit. Each file is a member of the library.

SAS data set
a logical data structure for SAS processing. There are two types of data sets: SAS data files and SAS data views. SAS data files contain data values in addition to descriptor information that is associated with the data. SAS data views contain only the descriptor information and information that is required for retrieving data values from other files.

SAS data view
a SAS data set that contains descriptor information but no data values. The data values are obtained from other files.

SAS file
a specially structured file that is created, organized, and maintained by the SAS System (optional). SAS files include SAS data sets, catalogs, and MDDBs.

SAS invocation
the process of calling or starting up SAS by an individual user through execution of the SAS command.

SAS log
a file that contains the SAS statements that you have submitted, messages about the execution of your program, and in some cases, output from the DATA step and from certain procedures.

SASProprietary
a fixed encoding algorithm that is included with Base SAS software and is available in all supported operating environments. It requires no additional SAS product licenses. The SASProprietary algorithm is strong enough to protect your data from casual viewing, but a determined hacker can break this encoding.

SAS/SECURE
an add-on product that provides additional encryption algorithms beyond the SAS proprietary algorithm. The encryption algorithms that are supported are: RC2, RC4, DES, and TripleDES. SAS/SECURE software requires a license and must be installed on each machine that runs a SAS/CONNECT or a SAS/SHARE server that will use the encryption algorithms.

SAS system option
an option that affects the processing of an entire SAS program or interactive SAS session from the time the option is specified until it is changed.

SAS windowing environment
an interactive, windowing interface to SAS software.

SCL
See SAS Component Language.

SCL function
in SCL, a mathematical relationship between variables that assigns exactly one value of the dependent variable to each combination of values of the independent variables. See also SAS Component Language (SCL).

script
an external file that is stored on the client that contains SAS script statements, which provide the instructions needed to establish and terminate the SAS/CONNECT link. Scripts are executed by the SIGNON and SIGNOFF commands.

script statement
a special SAS statement that is developed for use in scripts for SAS/CONNECT. Script statements are only used in scripts.

server
the side of the client/server relationship that receives requests from a "client" and responds to those requests by delivering a particular type of information. Access to computing resources and files (SAS files and external files) might be requested by a client. Implicit in the client/server relationship is the network. "Server" sometimes refers to the machine that the server runs on. See also client, SAS/CONNECT client, and SAS/CONNECT server.

server processing
the process of using communications software to process client tasks by using the CPU resources of a server. With SAS/CONNECT, the output and messages from a program that runs on the server are displayed on the client.

SMP hardware
a multi-processor machine where the processors share memory resources and are controlled by a single operating environment. See also MP(Multi–Processing) CONNECT.

spawner
a program that starts a SAS session on the server on behalf of the connecting client. Signing on to a SAS/CONNECT spawner that runs on a server is an alternative to signing on to a server by using a Telnet daemon. A spawner is assigned to a single port on the server. The port listens for requests for connection to the server.

SQL
See Structured Query Language (SQL).

SSL (Secure Sockets Layer)
a protocol that was developed by Netscape for transmitting private documents across the Internet. SSL uses a private key to encrypt data that is transmitted between a Web browser and a server.

statement label
a SAS name that is followed by a colon and that prefixes a statement in a DATA step. It enables statements to direct execution to that statement, bypassing other statements in the step.

Structured Query Language (SQL)
the standardized, high-level query language that is used in relational database management systems to create and manipulate database management system objects. SAS software implements SQL through the SQL procedure.

synchronization point
identifies the point during an asynchronous RSUBMIT at which the macro variable that is specified in the %SYSRPUT statement will be defined to the client session so that you can use it in your client processing.

synchronous processing
a type of server processing that requires you to submit one task at a time. You must wait until the server processing is complete before control is returned to the client session. Upon completion of the task in the server session, the results from synchronous processing are automatically transferred to the client session.

system option
See SAS system option.

TCP/IP (Transmission Control Protocol/ Internet Protocol)
a program-to-program interface that is supported on hardware from multiple vendors.

translation table
See character set translation table

upload
to copy a file from the client to the server. See also Data Transfer Services (DTS).

view
See SAS data view.

XMS (Cross-Memory Services)
an interface that is part of the z/OS operating environment and provides communication between SAS sessions that run in a single z/OS image.

Index

Your Turn

If you have comments or suggestions about the *SAS/CONNECT 9.1 User's Guide*, please send them to us on a photocopy of this page or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
email: **`yourturn@sas.com`**

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
email: **`suggest@sas.com`**